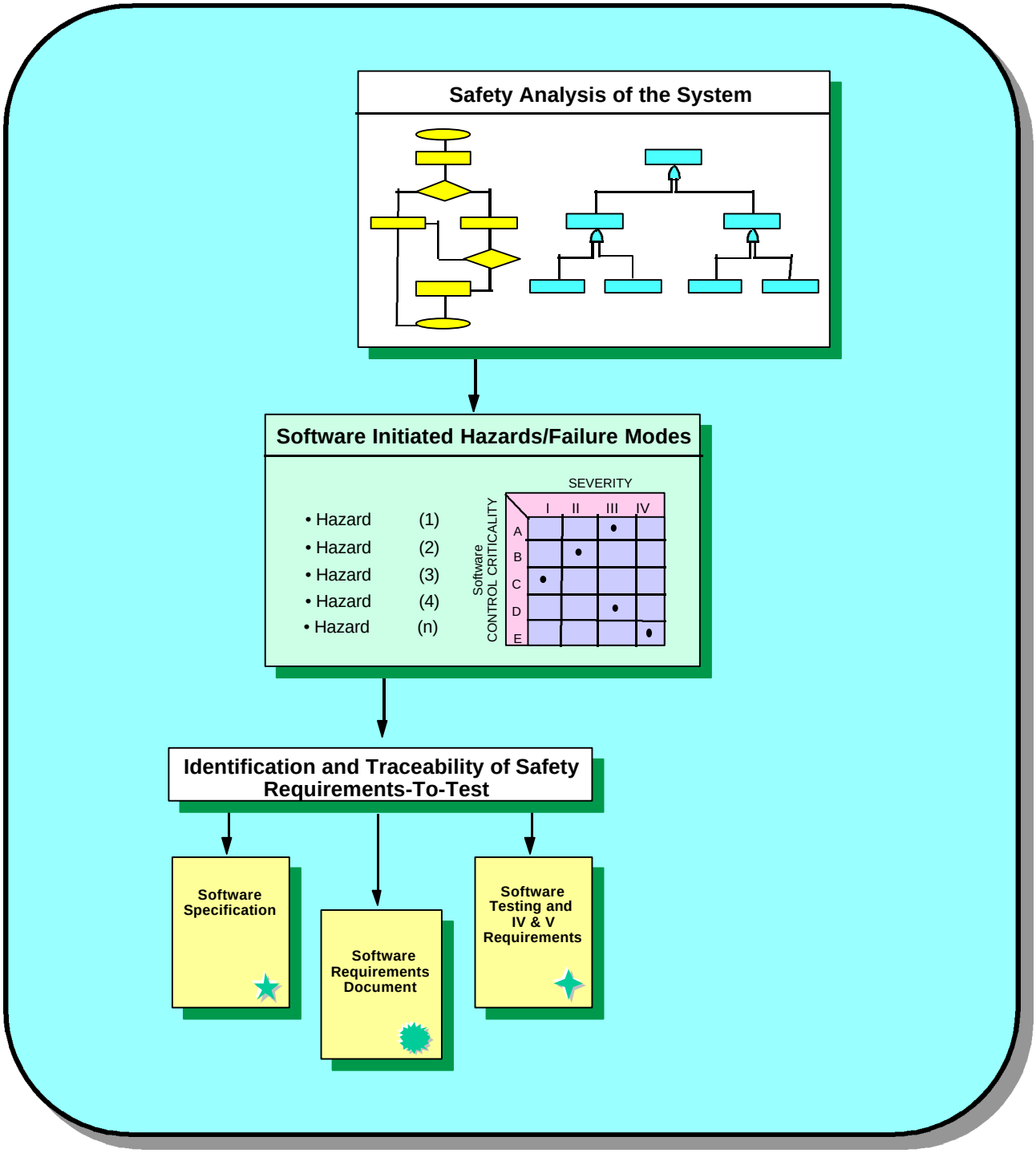
Joint Software System Safety Committee

SOFTWARE SYSTEM SAFETY HANDBOOK

A Technical and Managerial Team Approach

September 30, 1997



This Handbook
was funded and developed by the

Joint Services Computer Resources Management Group,
US Navy,
US Army,
and the US Air Force

Under the direction and guidance

Joint Services Software Safety Committee
of the
Joint Services System Safety Panel

Electronic Industries Association, G-48 Committee

Contributing Authors

John Bozarth
Michael Brown
Janet Gill
Steve Mattern

TABLE OF CONTENTS

1. EXECUTIVE OVERVIEW....1–1
2. INTRODUCTION TO THE HANDBOOK....2–1
2.1 INTRODUCTION....2–1
2.2 PURPOSE....2–2
2.3 SCOPE....2–2
2.4 AUTHORITY/STANDARDS....2–3
2.4.1 DEPARTMENT OF DEFENSE....2–3
2.4.1.1 DoD 5000.1....2–3
2.4.1.2 DoD 5000.2....2–4
2.4.1.3 Military standards....2–4
2.4.2 OTHER GOVERNMENT AGENCIES....2–8
2.4.2.1 Department of Transportation....2–8
2.4.2.2 NASA....2–10
2.4.3 COMMERCIAL....2–10
2.4.3.1 IEEE - 1228....2–10
2.4.3.2 EIA - 6B....2–10
2.4.3.3 IEC - 1508 (DRAFT)....2–11
2.5 HANDBOOK OVERVIEW....2–11
2.5.1 HISTORICAL BACKGROUND....2–11
2.5.2 PROBLEM IDENTIFICATION....2–12
2.5.2.1 Within System Safety....2–13
2.5.2.2 Within Software Development....2–15
2.5.3 MANAGEMENT RESPONSIBILITIES....2–15
2.5.4 INTRODUCTION TO THE “SYSTEMS” APPROACH....2–16
2.5.4.1 The Hardware Development Life cycle....2–17
2.5.4.2 The Software Development Life cycle....2–18
2.5.4.3 The Integration of Hardware and Software Life cycles....2–22
2.5.5 A “TEAM” SOLUTION....2–22
2.6 HANDBOOK ORGANIZATION....2–24
2.6.1 PLANNING....2–26
2.6.2 TASK IMPLEMENTATION....2–26
2.6.3 RISK ASSESSMENT AND ACCEPTANCE....2–26
2.6.4 SUPPLEMENTARY APPENDICES....2–27
3. INTRODUCTION TO RISK MANAGEMENT AND SYSTEM SAFETY....3–1
3.1 INTRODUCTION....3–1
3.2 A DISCUSSION OF RISK....3–1
3.3 TYPES OF RISK....3–2
3.4 AREAS OF PROGRAM RISK....3–3
3.4.1 SCHEDULE RISK....3–5
3.4.2 BUDGET RISK....3–6
3.4.3 SOCIAL-POLITICAL RISK....3–7
3.4.4 TECHNICAL RISK....3–7
3.5 SYSTEM SAFETY ENGINEERING....3–8
3.6 SAFETY RISK MANAGEMENT....3–11
3.6.1 INITIAL SAFETY RISK ASSESSMENT....3–11
3.6.1.1 Hazard and Failure Mode Identification....3–12
3.6.1.2 Hazard Severity....3–12
3.6.1.3 Hazard Probability....3–13
3.6.1.4 Hazard Risk Index (HRI) Matrix....3–14
3.6.2 SAFETY ORDER OF PRECEDENCE....3–15
3.6.3 ELIMINATION OR RISK REDUCTION....3–16
3.6.4 QUANTIFICATION OF RESIDUAL SAFETY RISK....3–17

3.6.5 MANAGING AND ASSUMING RESIDUAL SAFETY RISK....3–17
4. SOFTWARE SAFETY ENGINEERING....4–1
4.1 INTRODUCTION....4–1
4.1.1 SECTION FOUR FORMAT....4–3
4.1.2 PROCESS CHARTS....4–3
4.1.3 SOFTWARE SAFETY ENGINEERING PRODUCTS....4–4
4.2 SOFTWARE SAFETY PLANNING MANAGEMENT....4–5
4.2.1 PLANNING....4–6
4.2.1.1 Establish The System Safety Program....4–9
4.2.1.2 Defining Acceptable Levels of Risk....4–10
4.2.1.3 Program Interfaces....4–11
4.2.1.4 Contract Deliverables....4–15
4.2.1.5 Develop Software Hazard Criticality Matrix....4–16
4.2.2 MANAGING....4–19
4.3 SOFTWARE SAFETY TASK IMPLEMENTATION....4–23
4.3.1 SOFTWARE SAFETY PROGRAM MILESTONES....4–24
4.3.2 PRELIMINARY HAZARD LIST (PHL) DEVELOPMENT....4–27
4.3.3 TAILORING GENERIC SAFETY-CRITICAL REQUIREMENTS....4–29
4.3.4 PRELIMINARY HAZARD ANALYSIS....4–31
4.3.5 DERIVE SYSTEM SAFETY-CRITICAL SOFTWARE REQUIREMENTS....4–35
4.3.5.1 Preliminary Software Safety Requirements....4–37
4.3.5.2 Matured Software Safety Requirements....4–37
4.3.6 PRELIMINARY SOFTWARE DESIGN, SUBSYSTEM HAZARD ANALYSIS....4–38
4.3.6.1 Module Safety-criticality Analysis....4–41
4.3.6.2 Program Structure Analysis....4–42
4.3.6.3 Traceability Analysis....4–43
4.3.7 DETAILED SOFTWARE DESIGN SUBSYSTEM HAZARD ANALYSIS....4–44
4.3.7.1 Participate in Software Design Maturation....4–44
4.3.7.2 Detailed Design Software Safety Analysis....4–46
4.3.7.3 Detailed Design Analysis Related Sub-processes....4–49
4.3.8 SYSTEM HAZARD ANALYSIS....4–56
4.4 SOFTWARE SAFETY TESTING & RISK ASSESSMENT....4–60
4.4.1 SOFTWARE SAFETY TEST PLANNING....4–60
4.4.2 SOFTWARE SAFETY TEST ANALYSIS....4–61
4.4.3 SOFTWARE STANDARDS & CRITERIA ASSESSMENT....4–65
4.4.4 SOFTWARE SAFETY RESIDUAL RISK ASSESSMENT....4–67
4.5 MANAGING CHANGE....4–70
4.5.1 SOFTWARE CONFIGURATION CONTROL BOARD (CCB)....4–71
4.6 REUSABLE SOFTWARE....4–73
4.7 COTS SOFTWARE....4–73

LIST OF TABLES & FIGURES

Table 2-1 : Survey Response 2–14
Figure 2-1: Management Commitment to the Integrated Safety Process 2–16
Figure 2-2: Example of External System Interfaces 2–17
Figure 2-3: Weapon System life cycle....2–17
Figure 2-4: Relationship of Software to Hardware Development Life cycle....2–18
Figure 2-5: Waterfall, Grand Design Software Acquisition Model....2–20
Figure 2-6: Modified "V" Acquisition Life Cycle Model....2–20
Figure 2-7: Spiral Acquisition Life Cycle Model 2–22
Figure 2-8: Integration of Engineering Personnel and Processes....2–23
Figure 2-9: Handbook Layout....2–24
Figure 2-10: Section 4 Format....2–25
Figure 3-1: Types of Risk....3–3
Figure 3-2: Systems Engineering, Risk Management Documentation....3–6
Table 3-1: Hazard Severity....3–12
Table 3-2: Hazard Probability 3–13
Table 3-3: Hazard Risk Index....3–14
Figure 3-3: Hazard Reduction Order of Precedence 3–16
Figure 4-1: Section-4 Contents....4–1
Figure 4-2: Who is Responsible for System Software Safety?....4–2
Figure 4-3: Initial Process Chart Example....4–4
Figure 4-4: Software Safety Planning....4–5
Figure 4-5: Software Safety Planning by the Procuring agency....4–7
Figure 4-6: Software Safety Planning by the Developing Agency 4–8
Figure 4-7: Planning The Safety Criteria is Important....4–9
Figure 4-8: Risk Acceptance Matrix Example 4–11
Figure 4-9: Software Safety Program Interfaces....4–12
Figure 4-10: Ultimate Safety Responsibility 4–13
Figure 4-11: Proposed SSS Team Membership....4–14
Figure 4-12: Likelihood of Occurrence Example 4–17
Figure 4-13: Examples of Software Control Capabilities 4–18
Figure 4-14: Software Hazard Criticality Matrix -882C....4–19
Figure 4-15: Software Safety Program Management....4–20
Figure 4-16: Software Safety Task Implementation 4–23
Figure 4-17: Software Safety Program Schedule....4–25
Figure 4-18: PHL Development....4–27
Figure 4-19: Safety-critical Functions - An Example....4–28
Figure 4-20: Tailoring the Generic Safety Requirements 4–30
Figure 4-21: Example Generic Software Safety Requirements Tracking Worksheet....4–31
Figure 4-22: Preliminary Hazard Analysis 4–32
Table 4-1: Acquisition Process Trade-Off Analyses....4–33
Figure 4-23: Hazard Analysis Segment 4–33
Figure 4-24: PHA Hazard Control Record Example 4–35
Figure 4-25: Derive Safety-Specific Software Requirements....4–36
Figure 4-26: Software Safety Requirements Derivation 4–37
Figure 4-27: In-depth Hazard Cause Analysis 4–38
Figure 4-28: Preliminary Software Design Analysis 4–39
Figure 4-29. SSR Verification Tree 4–40
Table 4-3: Requirements Traceability Matrix Example....4–41
Table: 4-4: Safety-critical Function Matrix....4–42
Figure 4-30: Hierarchy Tree Example 4–42
Figure 4-31: Detailed Software Design Analysis 4–44
Figure 4-32. Verification Methods 4–45

Figure 4-33. Identification of Safety Related CSUs....4–46
Table 4-5: Data Item Example....4–50
Figure 4-34: Data Flow Diagram Example....4–51
Figure 4-35: Flow Chart Examples....4–52
Figure 4-36: System Hazard Analysis....4–56
Figure 4-37: SHA Interface Analysis Example....4–57
Figure 4-38: Documentation of Interface Hazards and Safety Requirements....4–58
Figure 4-39: Documenting Evidence of Hazard Mitigation....4–59
Figure 4-40: Software Safety Test Planning....4–60
Figure 4-41: Software Safety Test and Analysis....4–62
Figure 4-42: Software Requirements Verification....4–66
Figure 4-43: Residual Safety Risk Assessment....4–68
Figure 4-44: Generic Software Configuration Change Process....4–71

1. EXECUTIVE OVERVIEW

Since the development of the digital computer, software has continued to play an important and evolutionary role in the operation and control of hazardous, safety-critical functions. The reluctance of the engineering community to relinquish human control of hazardous operations has dramatically diminished in the last fifteen years. Today, digital computer systems have been given autonomous control over safety-critical functions in nearly every major technology in both commercial, and government systems. This revolution is primarily due to the ability of software to reliably perform critical control tasks at speeds unmatched by its human counterpart. Other factors influencing this transition is our ever-growing need and desire for increased versatility, greater performance capability, higher efficiency, and a decreased life cycle cost. In most instances, software can meet all of the above attributes of the systems performance when properly designed. The logic of the software allows for decisions to be implemented without emotion, and with speed and accuracy. This has forced the human operator out of the control loop because they can no longer keep pace with the speed, cost effectiveness, and decision making process of the system.

Therefore, there is a critical need to perform system safety engineering tasks on safety-critical systems to reduce the safety risk of all aspects of a program. These tasks includes the *software system safety activities* involving the design, code, test, IV&V, operation & maintenance, and change control functions of software engineering and development process.

The main objective (or definition) of system safety engineering, which includes software system safety, is:

"The application of engineering and management principles, criteria, and techniques to optimize all aspects of safety within the constraints of operational effectiveness, time, and cost throughout all phases of the system life cycle."

It must be noted that the ultimate responsibility for the development of a "safe system" rests with program management. The commitment to provide qualified people and an adequate budget and schedule for a software development program must begin with the program director or program manager. Top management must be a strong voice of safety advocacy and must communicate this personal commitment to each level of program and technical management. The program manager must be committed to support the integrated safety process between systems engineering, software engineering and safety engineering in the design, development, test, and operation of the system software.

Thus, the purpose of this Handbook is to:

Provide management and engineering, guidelines to achieve a reasonable level of assurance that software will execute within the system context with an acceptable level of safety risk.

2. INTRODUCTION TO THE HANDBOOK

2.1 INTRODUCTION

Section 2 of the Software System Safety Handbook (SSSH) should be read by all members of the Software System Safety (SSS) Team. This Section discusses the following major subjects:

- The major purpose for writing this Handbook
- The scope of the subject matter that the Handbook will present
- The authority by which an SSS program is conducted.
- Historical incidents that clearly demonstrate a need for SSS programs
- How this Handbook is organized and the best procedure for you to use to gain its full benefit.

You, as a member of the system safety team, are critical in the design, and redesign, of modern systems. Whether a hardware engineer, software engineer, "safety specialist", or manager, it is your responsibility, to ensure an acceptable level of safety is achieved and maintained throughout the life cycle of the system(s) you are helping develop. You can do this, and this Handbook will show you how through a rigorous and pragmatic application of software system safety planning and analysis.

Software system safety, an element of the total safety and software development program, cannot be allowed to function independently of the total effort. Nor can it be ignored. Systems, both "simple" and highly integrated multiple subsystems, are experiencing an extraordinary growth in the use of computers and software to monitor and/or control safety-critical subsystems or functions. A software specification error, design flaw, or the lack of generic safety-critical requirements e.g., no run-time error within these safety-critical subsystems, can contribute to or cause a system failure or erroneous human decision. Preventable death, injury, loss of the system or environmental damage can result. To achieve an acceptable level of safety for software used in critical applications, software safety engineering must be given primary emphasis early in the requirements definition and system conceptual design process. Safety-critical software must then receive continuous management emphasis and engineering analysis throughout the development and operational life cycles of the system.

This Software System Safety (SSS) Handbook is a joint effort. The U.S. Army, Navy, Air Force, and Coast Guard Safety Centers; in cooperation with the FAA, NASA, defense industry contractors and academia, are primary contributors. This extensive research captures the "best practices" pertaining to SSS program management and safety-critical software design. The Handbook consolidates these contributions into a single, user-friendly, resource. It aides the system safety team to understand their SSS responsibilities. By using the Handbook, you will appreciate the need for all disciplines to work together in identifying, controlling and managing software-related hazards within safety-critical components of hardware systems.

To summarize, the Handbook is a "how-to" guide for use in the understanding of both the complete SSS and the contribution of each functional discipline. It is applicable to all types of systems, in all types of operational uses.

2.2 PURPOSE

The purpose of the Software System Safety Handbook is to provide management and engineering, guidelines to achieve a reasonable level of assurance that software will execute within the system context with an acceptable level of safety risk.

2.3 SCOPE

This Handbook is a reference document and management tool to aid managers and engineers at all levels, and in any government and industry organization. It demonstrates "how to" develop and implement an effective SSS process. This process minimizes the likelihood or severity of system hazards caused by poorly specified, designed, or developed software in safety-critical applications.

The primary responsibility for management of the SSS process lies with the system safety manager/engineer in both the developer's (supplier) and acquirer's (customer) organization. However, nearly every functional discipline has a vital role and must be intimately involved in the SSS process. The SSS tasks, techniques, and process outlined in this Handbook are generic enough to be applied to any system which uses software in critical areas. It serves the need for all contributing disciplines to understand and apply qualitative and quantitative analysis techniques to ensure the safety of hardware systems controlled by software.

This Handbook is a guide and is not intended to supersede any Agency policy, standard, or guidance pertaining to system safety (MIL-STD-882C) or software engineering and development (MIL-STD-498). It is written to clarify the software system safety requirements and tasks specified in governmental and commercial standards and guideline documents. The Handbook is not a compliance document but a reference document. It provides the system safety manager and the software development manager with sufficient information to properly scope the SSS effort in the Statement of Work (SOW), to identify the data items needed to effectively monitor the contractor's compliance with the contract system safety requirements, and to evaluate his performance throughout the development life cycle. The Handbook is not a tutorial on software engineering. However, it does address some technical aspects of software function and design to assist with understanding software safety.

It is a Handbook objective to provide each member of the system software safety team with a basic understanding of sound systems and software safety practices, processes, and techniques. Another objective is to demonstrate the importance of each technical and managerial discipline to work hand-in-hand in defining software safety requirements for the safety-critical software components of the system. A final objective is to show how needed safety features can be designed into the software to eliminate or control identified hazards.

2.4 AUTHORITY/STANDARDS

Numerous directives, standards, regulations, and regulatory guidance establish the authority for system safety engineering requirements in the acquisition, development, and maintenance of software based systems. Although the primary focus of this handbook is targeted toward military systems, much of the authority for the establishment of DoD system safety, and software safety programs is derived from other governmental and commercial standards and guidance. We have documented many of these authoritative standards and guidelines within this handbook to first; establish their existence, second; to demonstrate the seriousness the government places on the reduction of safety risk for software performing safety-critical functions, and last; to consolidate in one place, all of the authoritative documentation a program manager, safety manager, or safety engineer would need to clearly demonstrate the mandated requirement and need for a software safety program to their superiors.

2.4.1 DEPARTMENT OF DEFENSE

Within the Department of Defense (DoD), and the acquisition corps of each branch of military service, the primary documents of interest pertaining to system safety and software development include; DoD 5000.1, Defense Acquisition; DoD 5000.2, Defense Acquisition Management, Policies and Procedures; MIL-STD-498, Software Development and Documentation; and MIL-STD-882C, System Safety Program Requirements. The authority of the acquisition professional to establish a software safety program is provided in the following paragraphs. These paragraphs are quoted or summarized from various DoD directives and military standards to clearly define the mandated requirement for all DoD systems acquisition and development programs to incorporate safety requirements and analysis into the design, development, testing, and support of software being used to perform or control critical system functions. The DoD documents also levy the authority and responsibility for establishing and managing an effective software safety program to the highest level of program authority.

2.4.1.1 DOD 5000.1

DoD 5000.1, Defense Acquisition, February 23, 1991; Part 1: A. Overview, #2, establishes the requirement and need for and aggressive risk management program for acquiring quality products. In addition Part 1: C., Acquiring Quality Products, establishes the requirement for acquisition strategies and objectives to control risk, including safety risk.

- **Overview, 2. Acquiring Quality Products.** A rigorous, event-oriented, management process shall be used for acquiring quality products that emphasize effective acquisition planning, improved communications with the users, and aggressive risk management by both government and industry.
- **1. C. Acquiring Quality Products (b):** Program plans must provide for a systems engineering approach to the simultaneous design of the product and its associated manufacturing, test, and support processes. This concurrent engineering approach is essential to achieving a careful balance among system design requirements (e.g., operational performance, production, reliability, maintainability, logistics and human factors engineering, safety, survivability, interoperability, and standardization).

2.4.1.2 DOD 5000.2

DoD 5000.2, Defense Acquisition Management Policies and Procedures, February 26, 1993, Part 6, Section D, "*Computer Resources*", establishes the interface between system safety engineering and software development.

- **Attachment 1, Software Engineering Practices, Section 2, Follow a Disciplined Process, (b.)**: "Software system safety techniques, analyses, and approaches described in MIL-STD-882 should be used to ensure the system safety process supports the DOD-STD-2167 (superseded by MIL-STD-498) software development process ...."

Part 6, Section I, *"System Safety, Health Hazards and Environmental Impact"*, establishes the requirement for system safety engineering on DoD system developments. Selected text of Part 6, Section I is provided which establishes and supports the system safety and software safety engineering associated with the design, development, test, and deployment of a system.

- **Paragraph 1, Purpose, (b.):** "These policies and procedures establish the basis for effectively integrating system safety, health hazard, and environmental considerations into the system engineering process."
- **Paragraph 2, Policies, (a.):** "Scientific and engineering principles shall be applied during design and development to identify and reduce hazards associated with system operation and support with the objective of designing the safest possible systems consistent with mission requirements and cost effectiveness."
- **Paragraph 2, Policies, (a.)(1):** "Appropriate system safety and health hazard objectives shall be established early in the program and used to guide system safety and health hazard activities and the decision process."
- **Paragraph 2, Policies, (c.):** "System safety engineering programs shall be designed to work in harmony with the other comprehensive DoD product improvement programs (e.g., manpower, personnel, and ...software quality assurance programs)."
- **Paragraph 3, Procedures, (a.):** "System Safety. A system safety program that identifies, evaluates, and eliminates or controls system hazards will be established through the tailored application of MIL-STD-882 ..., adapted to specific program characteristics."
- **Paragraph 3, Procedures, (a.)(1):** "The total system, including hardware, software, testing, manufacture, and support, will be evaluated for known or potential hazards for the entire life cycle."

2.4.1.3 MILITARY STANDARDS

2.4.1.3.1 MIL-STD 882B

MIL-STD 882B, System Safety Program Requirements, March 30, 1984, remains on numerous government programs which were contracted during the 1980's prior to the issuance of MIL-STD 882C. The objective of this standard is the establishment of a system safety program to ensure safety, consistent with mission requirements, is designed into systems, subsystems, equipment,

and facilities, and their interfaces. Authors of this standard recognized the safety risk influence software presented in safety-critical systems. The standard provides guidance and specific tasks for the development team to address the software, hardware, system and human interfaces. These include the 300-series tasks. The purpose of each task is as follows:

- **Task 301, Software Requirements Hazard Analysis**: The Purpose of Task 301 is to require the contractor to perform and document a software requirements hazard analysis. The contractor shall examine system and software requirements and design in order to identify unsafe modes for resolution, such as out-of-sequence, wrong event, inappropriate magnitude, inadvertent command, adverse environment, deadlocking, failure-to-command, etc. The analysis shall examine safety-critical computer software components at a gross level to obtain an initial safety evaluation of the software system.

- **Task 302, Top-Level Design Hazard Analysis**: The purpose of Task 302 is to require the contractor to perform and document a Top-Level Hazard Analysis. The contractor shall analyze the top-level design, using the results of the safety requirements hazard analysis if previously accomplished. This analysis shall include the definition and subsequent analysis of safety-critical computer software components, identifying the degree of risk involved, and the design and test plan to be implemented. The analysis shall be shall be substantially complete before the software detailed design is started. The results of the analysis shall be present at the preliminary design review.

- **Task 303, Detailed Design Hazard Analysis**: The purpose of Task 303 is to require the contractor to perform and document a Detailed Design Hazard Analysis. The contractor shall analyze the software detailed design using the results of the Software Requirements Hazard Analysis and the Top-Level Design Hazard Analysis to verify the correct incorporation of safety requirements and to analyze the safety-critical computer software components. This analysis shall be substantially complete before coding of the software is started. The results of the analysis shall be presented at the critical design review.

- **Task 304, Code-Level Software Hazard Analysis**: The purpose of Task 304 is to require the contractor to perform and document a Code-Level Software Hazard Analysis. Using the results of the Detailed Design Hazard Analysis, the contractor shall analyze program code and system interfaces for events, faults, and conditions which could cause or contribute to undesired events affecting safety. This analysis shall start when coding begins, and shall be continue throughout the system life cycle.

- **Task 305, Software Safety Testing**: The purpose of Task 305 is to require the contractor to perform and document software safety testing to ensure that all hazards have been eliminated or controlled to an acceptable level of risk.

- **Task 306, Software/User Interface Analysis**: The purpose of Task 306 is to require the contractor to perform and document a Software/User Interface Analysis and the development of Software Users Procedures.

- **Task 307, Software Change Hazard Analysis**: The purpose of Task 307 is to require the contractor to perform and document a Software Change Hazard Analysis. The contractor shall analyze all changes, modifications, and patches made to the software for safety hazards.

2.4.1.3.2 MIL-STD 882

MIL-STD-882C, System Safety Program Requirements, January 19, 1993, establishes the requirement for a detailed system safety engineering and management activities on all system procurements within the DoD. This includes the integration of software safety, within the context of the system safety program. Although MIL-STD 882B remains on older contracts within the DoD, MIL-STD 882C is the current system safety standard as of the date of this handbook.

- **Paragraph 4, General Requirements, 4.1, System Safety Program:** "The contractor shall establish and maintain a system safety program to support efficient and effective achievement of overall system safety objectives."
- **Paragraph 4.2, System Safety Objectives:** "The system safety program shall define a systematic approach to make sure that:...(b.) Hazards associated with each system are identified, tracked, evaluated, and eliminated, or the associated risk reduced to a level acceptable to the MA throughout entire life cycle of a system." [MA = Managing Authority]
- **Paragraph 4.3, System Safety Design Requirements:** "...Some general system safety design requirements are:...(j.) Design software controlled or monitored functions to minimize initiation of hazardous events or mishaps."
- **Task 202, Preliminary Hazard Analysis, Section 202.2, Task Description:** "...The PHA shall consider the following for identification and evaluation of hazards as a minimum: (b.) Safety related interface considerations among various elements of the system (e.g., material compatibility's, electromagnetic interference, inadvertent activation, fire/explosive initiation and propagation, and hardware and software controls.) This shall include consideration of the potential contribution by software (including software developed by other contractors/sources) to subsystem/system mishaps. Safety design criteria to control safety-critical software commands and responses (e.g., inadvertent command, failure to command, untimely command or responses, inappropriate magnitude, or (MA)-designated undesired events) shall be identified and appropriate actions taken to incorporate them in the software (and related hardware) specifications."

Task 202 is included as a representative description of tasks integrating software safety. The general description is also applicable to all the other tasks specified in MIL-STD-882C. The point is that software safety must be an integral part of system safety and software development.

2.4.1.3.3 DOD-STD 2167A

Although this standard has been currently replace by MIL-STD 498, DoD-STD 2167A, Military Standard Defense System Software Development, February 29, 1988, remains on numerous older

contracts within the DoD. This standard establishes the uniform requirements for software development that are applicable throughout the system life cycle. The requirements of this standard provide the basis for Government insight into a contractor's software development, testing, and evaluation efforts. The specific requirement of the standard which establishes a system safety interface with the software development process is:

- **Paragraph 4.2.3, Safety Analysis**: The contractor shall perform the analysis necessary to ensure that the software requirements, design, and operating procedures minimize the potential for hazardous conditions during the operational mission. Any potentially hazardous conditions or operating procedures shall be clearly defined and documented.

2.4.1.3.4 MIL-STD-498

MIL-STD-498, Software Development and Documentation, December 5, 1994, Paragraph 4.2.4.1, establishes an interface with system safety engineering and defines the safety activities which are required for incorporation into the software development throughout the acquisition life cycle. This standard merges DOD-STD-2176A and DOD-STD-7935A to define a set of activities and documentation suitable for the development of both weapon systems and automated information systems. Other changes include improved compatibility with incremental and evolutionary development models; improved compatibility with non-hierarchical design methods; improved compatibility with computer-aided software engineering (CASE) tools; alternatives to, and more flexibility in, preparing documents; clearer requirements for incorporating reusable software; introduction of software management indicators; added emphasis on software support; and improved links to systems engineering. This standard can be applied in any phase of the system life cycle.

- **Paragraph 4.2.4.1, Safety Assurance:** "The developer shall identify as safety-critical those computer software configuration items (CSCI's) or portions thereof whose failure could lead to a hazardous system state (one that could result in unintended death, injury, loss of property, or environmental harm). If there is such software, the developer shall develop a safety assurance strategy, including both tests and analyses, to assure that the requirements, design, implementation, and operating procedures for the identified software minimize or eliminate the potential for hazardous conditions. The strategy shall include a software safety program which shall be integrated with the system safety program if one exists. The developer shall record the strategy in the software development plan, implement the strategy, and produce evidence, as part of required software products, that the safety assurance strategy has been carried out."

In the case of reusable software products (this includes COTS), MIL-STD-498 states:

- **Appendix B, B.3, Evaluating Reusable Software Products, (b.):** "General criteria shall be the software product's ability to meet specified requirements and to be cost effective over the life of the system. Non-mandatory examples of specific criteria include, but are not limited to:...b. Ability to provide required safety, security, and privacy."

2.4.2 OTHER GOVERNMENT AGENCIES

Outside the DoD, other governmental agencies are not only interested in the development of safe software, but are aggressively pursuing the development or adoption of new regulations, standards, and guidance for establishing and implementing software system safety programs for their developing systems. Those governmental agencies expressing an interest and actively participating in the development of this handbook are identified below. Also included is the authoritative documentation used by these agencies which establish the requirement for a software system safety program.

2.4.2.1 DEPARTMENT OF TRANSPORTATION

2.4.2.1.1 FEDERAL AVIATION AUTHORITY - (FAA)

FAA Order 1810 "ACQUISITION POLICY" establishes general policies and the framework for acquisition for all programs that require operational or support needs for the Federal Aviation Administration (FAA). It implements the Department of Transportation (DOT) Major Acquisition Policy and Procedures (MAPP) in its entirety and consolidates the contents of more than 140 FAA Orders, standards, and other references. FAA Order 8000.70 "FAA SYSTEM SAFETY PROGRAM" (SSP) requires that the FAA SSP be used, where applicable, to enhance the effectiveness of FAA safety efforts through the uniform approach of system safety management and engineering principles and practices."[1]

A significant FAA safety document is (RTCA)/DO-178B, Software Considerations In Airborne Systems and Equipment Certification. Important points from this resource are:

- **Paragraph 1.1, Purpose**: "The purpose of this document is to provide guidelines for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements."
- **Paragraph 2.1.1, Information Flow from System Processes to Software Processes**: "The system safety assessment process determines and categorizes the failure conditions of the system. Within the system safety assessment process, an analysis of the system design defines safety related requirements that specify the desired immunity from, and system responses to, these failure conditions. These requirements are defined for hardware and software to preclude or limit the effects of faults, and may provide fault detection and fault tolerance. As decisions are being made during the hardware design process and software development processes, the system safety assessment process analyzes the resulting system design to verify that it satisfies the safety-related requirements.

The safety-related requirements are inputs to the software life cycle process. To ensure that they are properly implemented, the system requirements typically include or reference:

- The system description and hardware definition.

[1] FAA System Safety Handbook, Draft, December 31, 1993

- Certification requirements, including Federal Aviation Regulation (FAR-United States), Joint Aviation Regulations (JAR-Europe), Advisory Circulars (United States), etc.
- System requirements allocated to software, including functional requirements, performance requirements, and safety-related requirements.
- Software level(s) and data substantiating their determination, failure conditions, their Hazard Risk Index (HRI) categories, and related functions allocated to software.
- Software strategies and design constraints, including design methods, such as, partitioning, dissimilarity, redundancy, or safety monitoring.
- If the system is a component of another system, the safety-related requirements and failure conditions for that system.

System life cycle processes may specify requirements for software life cycle processes to aid system verification activities.

2.4.2.1.2 COAST GUARD

COMDTINST M41150.2D, *Systems Acquisition Manual*, December 27, 1994, or the "SAM" establishes policy, procedures, and guidance for the administration of Coast Guard major acquisition projects. The SAM implements the Department of Transportation (DOT) Major Acquisition Policy and Procedures (MAPP) in it's entirety. The "System Safety Planning" section of the SAM requires the use of MIL-STD 882C in all Level I, IIIA, and IV acquisitions. The SAM also outlines system hardware and software requirements and concerns in the "Integrated Logistics Support Planning" section of the Manual.

Using MIL-STD 498 as a foundation, the Coast Guard has developed *a "Software Development and Documentation Standards*, Draft, May 1995" document for internal Coast Guard use. The important points from this document are:

- **Paragraph 1.1, Purpose**: "The purpose of this standard is to establish Coast Guard software development and documentation requirements to be applied during the acquisition, development, or support of the software system.
- **Paragraph 1.2, Application**: "This standard is designed to be contract specific applying to both contractors or any other government agency(s) who would develop software for the Coast Guard."
- **Paragraph 1.2.3, Safety Analysis**: "Safety shall be a principle concern in the design and development of the system and it's associated software development products." This standard will require contractors to develop a software safety program, integrating it with the system safety program. This standard also requires the contractor to perform safety analysis on software to identify, minimize, or eliminate hazardous conditions that could potentially affect operational mission readiness.

2.4.2.2 NASA

The National Aeronautics and Space Administration (NASA), has been developing safety-critical, software-intensive aeronautical and space systems for many years. To support the required planning of software safety activities on these research and operational procurements, NASA published NSS 1740.13, Interim, Software Safety Standard, in June 1994. "The purpose of this standard is to provide requirements to implement a systematic approach to software safety as an integral part of the overall system safety programs. It describes the activities necessary to ensure that safety is designed into software that is acquired or developed by NASA and that safety is maintained throughout the software life cycle." The development of this NASA standard was influenced by several DoD and Military Standards including both DOD-STD-2167A, *Defense System Software Development*, and MIL-STD-882C, *System Safety Program Requirements.*

The defined purpose of NSS 1740.13, is to ensure that software does not cause or contribute to a system reaching a hazardous state; that it does not fail to detect or take corrective action if the system reaches a hazardous state; and that it does not fail to mitigate damage if an accident occurs.

2.4.3 COMMERCIAL

Unlike the historical relationship established between DoD agencies' and their contractors, whereby the services contractually mandated the contractors adherence to military standards requirements and tasks for system safety, commercial companies are not obligated to a specified, quantifiable level of safety risk management on the products they produce (unless contractually obligated through a subcontract arrangement with another company or agency). Instead, they are primarily motivated by economical, ethical, and legal liability factors. For those commercial companies that are motivated or compelled to pursue the elimination or control of safety risk in software, several commercial standards are available to provide them guidance. This handbook will only reference a few of the most popular. While these commercial standards are readily accessible, few provide the practitioner with a defined software safety process or the "how-to" guidance required to implement the process.

2.4.3.1 IEEE - 1228

The Institute of Electrical and Electronic Engineers (IEEE) published IEEE Std 1228-1994, IEEE Standard for Software Safety Plans, for the purpose of describing the minimum acceptable requirements for the content of a software safety plan. This standard contains four clauses. Clause 1 discusses the application of the standard. Clause 2 lists references to other standards. Clause 3 provides a set of definitions and acronyms used in the standard. Clause 4 contains the required content of a software safety plan. An informative annex is included and discusses software safety analyses. IEEE-1228 is intended to be "wholly voluntary" and was written for those who are responsible for defining, planning, implementing, or supporting software safety plans. This standard closely follows the methodology of MIL-STD-882B, Change Notice 1.

2.4.3.2 EIA - 6B

The Electronic Industries Association (EIA), G-48 System Safety Committee published the Safety Engineering Bulletin No. 6B, System Safety Engineering In Software Development, in 1990. The

System Safety G-48 Committee has as its interest, the procedures, methodology and development of criteria for the application of system safety engineering to systems, subsystems, and equipment. The purpose of the document is "...to provide guidelines on how a system safety analysis and evaluation program should be conducted for systems which include computer-controlled or -monitored functions. It addresses the problems and concerns associated with such a program, the processes to be followed, the tasks which must be performed, and some methods which can be used to effectively perform those tasks."

2.4.3.3 IEC - 1508 (DRAFT)

The International Electrotechnical Commission (IEC) has submitted a draft international standard which is primarily concerned with safety-related control systems incorporating electrical/electronic/programmable electronic devices. It also provides a framework which is applicable to safety-related systems irrespective of the technology on which those systems are based (e.g., mechanical, hydraulic, or pneumatic). "The draft International Standard has two concepts which are fundamental to its application - namely, a **Safety Life cycle** and **Safety Integrity Levels**. The **Overall Safety Life Cycle** is introduced in Part 1 and forms the central framework which links together most of the concepts in this draft International Standard."[2]

This International Standard consists of seven parts:

Part 1: General Requirements

Part 2: Requirements for electrical/electronic/programmable electronic systems (E/E/PES)

Part 3: Software Requirements

Part 4: Definitions

Part 5: Guidelines on the application of Part 1

Part 6: Guidelines on the application of Part 2 and Part 3

Part 7: Bibliography of techniques

The draft standard addresses all relevant safety life cycle phases when E/E/PES's are used to perform safety functions. It has been developed with a rapidly developing technology in mind. The framework in this standard is considered to be sufficiently robust and comprehensive to cater for future developments.

2.5 HANDBOOK OVERVIEW

2.5.1 HISTORICAL BACKGROUND

Since the development of the digital computer, software has continued to play an important and evolutionary role in the operation and control of hazardous, safety-critical functions. The reluctance of the engineering community to relinquish human control of hazardous operations has dramatically diminished in the last fifteen years. Today, digital computer systems have been given autonomous control over safety-critical functions in nearly every major technology in both

[2] IEC 1508-1, Ed. 1, (DRAFT), Functional Safety; Safety Related Systems, June 1995

commercial, and government systems. This revolution is primarily due to the ability of software to reliably perform critical control tasks at speeds unmatched by its human counterpart. Other factors influencing this transition is our ever-growing need (or desire) for increased versatility, greater performance capability, higher efficiency, and a decreased life cycle cost (as it is easier to change software than hardware). In most instances, software can meet all of the above attributes of the systems performance when properly designed. The logic of the software allows for decisions to be implemented without emotion, and with speed and accuracy. This has forced the human operator out of the control loop because they can no longer keep pace with the speed, cost effectiveness, and decision making process of the system.

The introduction of software-controlled, safety-critical systems has caused considerable ramifications in the managerial, technical, safety, economic, and scheduling risks of both hardware and software system developments. Although this risk is discussed extensively in Section 3, the primary focus of this handbook is the identification, documentation (to include evidence through analyses) and elimination, or control, of the safety risk associated with software in the design, development, test, operation and support of the "system".

A software design flaw or run-time error within safety-critical functions of a system introduces the potential of a hazardous condition or failure mode which could result in death, personal injury, loss of the system or environmental damage. Appendix E. provides abstracts of numerous examples of software-influenced accidents and failures. These appendices include:

E.1 - Missile launch timing error causes hang-fire.

E.2 - Drone lost from incomplete requirements for flight control sequencing.

E.3 - Reused software causes flight controls shut down.

E.4 - Software-configurable flight controls become unstable during test of new flight envelope.

E.5 - Flight controls fail at supersonic transition.

E.6 - Software control of weapon lost due to loss of data from electromagnetic interference.

E.7 - Incorrect missile firing due to invalid setup sequence.

E.8 - Operator choice of weapon release over-ridden by software control.

2.5.2 PROBLEM IDENTIFICATION

Since the introduction of digital controls, the engineering community has wrestled (along with their research brethren) with processes, methods, techniques, and tools for the sole purpose of reducing the safety risk of software controlled operations. Each engineering discipline viewed the problem from a vantage point and perspective from within the confines of their respective area of expertise. In many instances, this view was analogous to the view seen when looking down a tunnel. The responsibilities of, and the interfaces with, other management and engineering functions were often distorted due to individual or organizational biases.

Part of the problem is that SSS is still a relatively new discipline and methodologies, techniques, and processes are still being researched and evaluated in terms of logic and practicality on

software development activities. As with any new discipline, the problem must be adequately defined prior to the application of recommend practices.

2.5.2.1 WITHIN SYSTEM SAFETY

From the perspective of most of the system safety community, digital control of safety-significant functions introduced a new and unwanted level of uncertainty to a historically sound hazards analysis methodology for hardware. Many within system safety were unsure of how to integrate software into the system safety process, techniques, and methods which were currently being used. System safety managers and engineers, educated in the 1950's, 60's, and 70's, had relatively no computer-, or software-related education or experience. This compounded their reluctance to, or in many cases, their desire or ability to even address the problem.

In the late 1970's and early 1980's, bold individuals within the safety, software, and research (academia) communities took their first steps in identifying and addressing the safety risks associated with software. Although these individuals may not have been in total lock-step and agreement, they did, in fact, lay the necessary foundation for where we are today. It was during this period that MIL-STD-882B was developed and published. This was the first military standard to require software system safety engineering and management activities and tasks be performed by the developing contractor. However, due to the distinct lack of cooperation or communication between the system safety and software engineering disciplines in defining a workable process for identifying and controlling software-related hazards in developing systems, the majority of system safety professionals waited for academia or the software engineering community to develop a "silver bullet" analysis methodology or tool. It was their hope that such an analysis technique or verification tool could be applied to finished software code to identify any fault paths to hazard conditions or failure modes which could then be quickly corrected prior to delivery. This concept did not include the identification of system hazard and failure modes caused (or influenced) by software inputs, or the identification of safety-specific requirements to mitigate these hazards and failure modes. Note that there is yet no "silver bullet", and there will probably never be one. Even if a silver bullet existed, it would be used too late in the system development life cycle to influence design.

Software Hazard Analysis Tools

No.	Tool/Technique	No.	Tool/Technique
8	Fault Tree Analysis	1	Hierarchy Tool
4	Software PrelimHazard Analysis	1	Compare & Certification Tool
3	Traceability Analysis	1	System Cross Check Matrices
3	Failure Modes & Effects Analysis	1	Top-Down Review of Code
2	Requirements Modeling/Analysis	1	Software Matrices
2	Source Code Analysis	1	Thread Analysis
2	Test Coverage Analysis	1	Petri-Net Analysis
2	Cross Reference Tools	1	Sofware Hazard List
2	Code/Module Walkthrough	1	BIT/FIT Plan
2	Sneak Circuit Analysis	1	Nuclear Safety Cross-Check Anal.
2	Emulation	1	Mathematical Proof
2	SubSystem Hazard Analysis	1	Software Fault Hazard Analysis
1	Failure Mode Analysis	1	MIL-STD 882B, Series 300 Tasks
1	Prototyping	1	Topological Network Trees
1	Design and Code Inspections	1	Critical Function Flows
1	Checklist of Common SW Errors	1	Black Magic
1	Data Flow Techniques		

NOTE: No. = Number of times the response was provided from those responding

Table 2-1 : Survey Response

To further obscure the issue, the safety community within the DoD finally recognized that contractors developing complex hardware and software systems must perform “software safety tasks”. As a result contracts from that point forward included tasks to include software in the system safety process. The contractor was now forced to propose, bid, and perform software safety tasks with relatively little guidance. Those with software safety tasks on contract were in a desperate search for *any* tool, technique, or method which would assist them in meeting their contractual requirements. This was demonstrated by a sample population survey conducted in 1988 involving software and safety engineers and managers. When these professionals were asked to identify the tools and techniques which they used to perform contractual obligations pertaining to software safety, they provided answers which were wide and varied across the analysis spectrum. Of 148 surveyed, 74 provided responses. These answers are provided in Table 2-1. It is interesting to note that of all respondents to the survey, only five percent felt they had accomplished anything meaningful in terms of reducing the safety risk of the software analyzed.

The information provided in Table 2-1 demonstrated the lack of any standardized approach for the accomplishment of software safety tasks which were levied contractually. It also appeared as if the safety engineer either tried to accomplish the required tasks using a standard system safety approach, or borrowed the most logical tool available from the software development group. In either case, they remained unconvinced of their efforts’ utility in reducing the safety risk of the software performing in their system.

2.5.2.2 WITHIN SOFTWARE DEVELOPMENT

Historically, the software development and engineering community made about as much progress addressing the software safety issue as did system safety. Although most software development managers recognized the safety risk potential that the software posed within their systems, few possessed the credible means or methods for both minimizing the risk potential, and verifying that safety specification requirements had been achieved in the design. Most failed to include system safety engineering in software design and development activities, and most did not recognize that this interface was either needed or required.

A problem, which still exists today, is that most educational institutions do not teach students in computer science and software engineering that there is a required interface with safety engineering when software is integrated into a potentially hazardous system. Although the software engineer may implement a combination of fault avoidance, fault removal, and/or fault tolerance techniques in the design, code, or test of software, they usually failed to tie the fault or error potential to a specific system hazard or failure mode. While these efforts most likely increase the overall reliability of the software, many fail to verify that the safety requirements of the system have been implemented to an acceptable level.

It is essential that the software development community understand the needed interface with system safety and that system safety understand their essential interface with software development.

2.5.3 MANAGEMENT RESPONSIBILITIES

The ultimate responsibility for the development of a “safe system” rests with program management. The commitment of qualified people and an adequate budget and schedule for a software development program must begin with the program director or program manager. Top management must be a strong voice of safety advocacy and must communicate this personal commitment to each level of program and technical management. The program manager must be committed to support the integrated safety process and information between systems engineering, software engineering and safety engineering in the design, development, test, and operation of the system software. Figure 2-1 graphically portrays the management element into the integrated team.

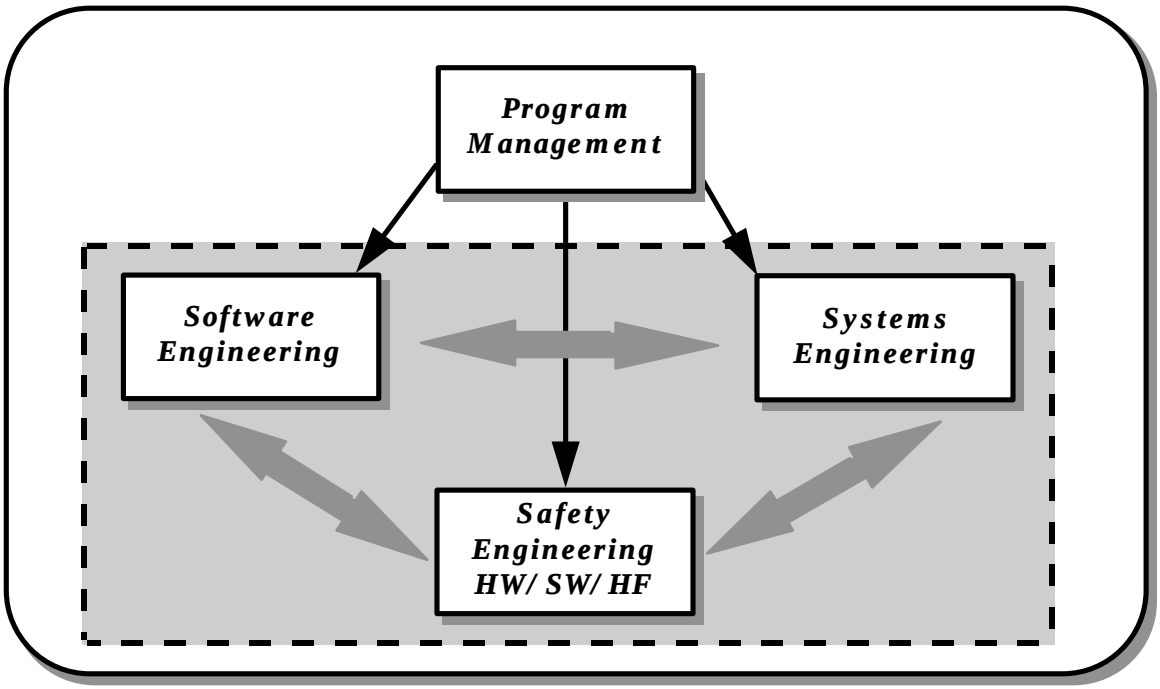


Figure 2-1: Management Commitment to the Integrated Safety Process

2.5.4 INTRODUCTION TO THE "SYSTEMS" APPROACH

System safety engineering has historically demonstrated the benefits of a "systems" approach to safety risk analysis and mitigation. When a hazard analysis is conducted on a hardware subsystem as a separate entity, it will produce a set of unique hazards applicable only to that subsystem. However, when that same subsystem is analyzed in the context of its physical, functional, and zonal interfaces with the rest of the "system components", the analysis will likely produce numerous other hazards which were not discovered by the original analysis. Conversely, the results of a system analysis may demonstrate that hazards identified in the subsystem analysis were either reduced or eliminated by other components of the system. Regardless, the identification of critical subsystem interfaces (such as software) with their associated hazards is a vital aspect of safety risk minimization for the total system.

When analyzing software which performs safety-critical functions within a system, a "systems approach" is also required. The success of a software safety program is predicated on it! Today's software is a very critical component of the safety risk potential of systems being developed and fielded. Not only are the internal interfaces of the system important to safety, but so are the external interfaces.

Figure 2-2 depicts specific software internal interfaces within the "system" block (within the ovals) and also external software interfaces to the system. Each identified software interface may possess safety risk potential to the operators, maintainers, environment, or the system itself. The acquisition and development process must consider these interfaces during the design of both the hardware and software systems. To accomplish this, the hardware and software development life cycles must be fully understood and integrated by the design team.

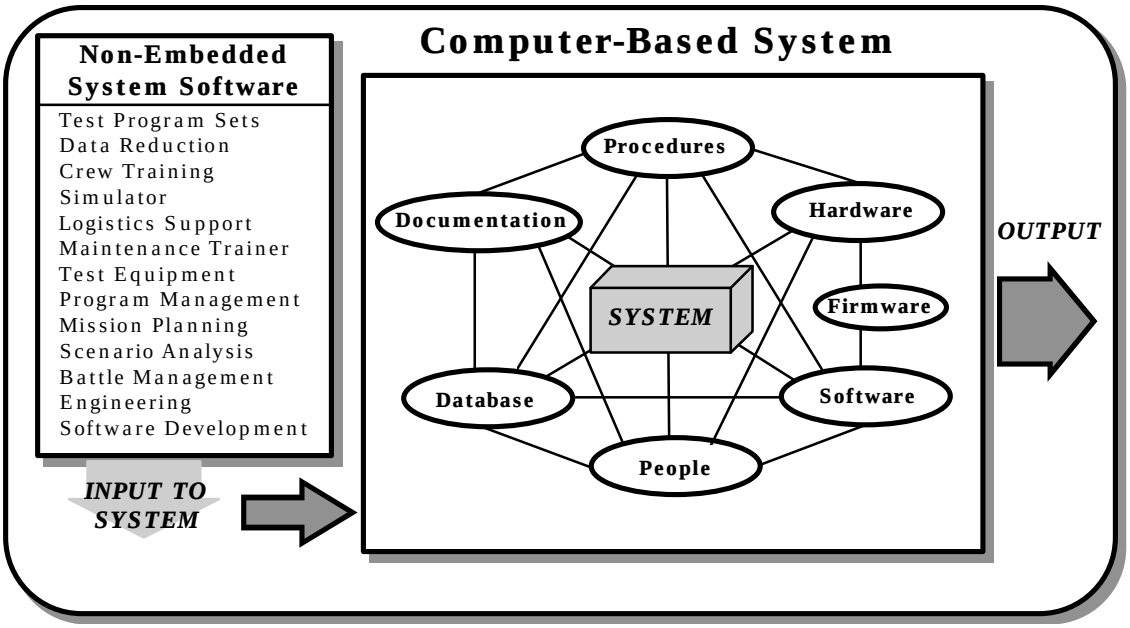


Figure 2-2: Example of External System Interfaces

2.5.4.1 THE HARDWARE DEVELOPMENT LIFE CYCLE

The typical hardware development life cycle has been in existence for many years. It is a proven acquisition model which has produced, in most instances, the desired engineering results in the design, development, manufacturing, fabrication, and test activities. It consists of five phases. These are identified as the concept exploration and definition, demonstration and validation, engineering and manufacturing development, production and deployment, and operations and support phases. Each phase of the life cycle ends, and the next phase begins, with a milestone decision point (0, I, II, III, and IV). An assessment of the system design and program status is made at each milestone decision point and plans are made or reviewed for subsequent phases of the life cycle. Specific activities conducted for each milestone decision is covered in numerous system acquisition management courses and documents. Therefore, they will not be discussed in greater detail in the contents of this Handbook.

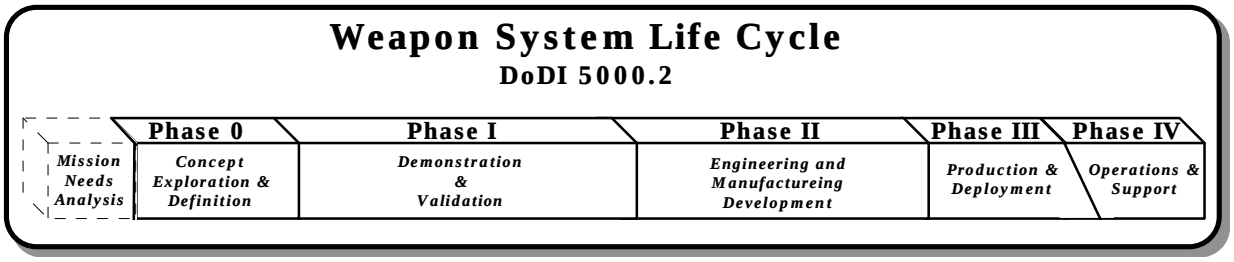


Figure 2-3: Weapon System life cycle

The purpose of introducing the system life cycle in this handbook is to familiarize the reader with a typical life cycle model. The one shown in Figure 2-3 is used in most DoD procurements. It identifies and establishes defined phases for the development life cycle of a system and can be overlaid on a proposed timetable to establish a milestone schedule. Detailed information

regarding milestones and phases of a system life cycle can be obtained from Defense Systems Management College (DSMC) documentation, and systems acquisition management course documentation of the individual services.

2.5.4.2 THE SOFTWARE DEVELOPMENT LIFE CYCLE

The system safety team must be fully aware of the software life cycle being used by the development activity. In the past several years, numerous life cycle models have been identified, modified, and used in some capacity on a variety of software development programs. This Handbook will not enter into a discussion as to the merits and limitations of different life cycle process models because the decision for or against a model must be made by the software engineering team for an individual procurement. The important issue here is for the system safety team to recognize which model is being used and how they should correlate and integrate safety activities with the chosen software development model to achieve the desired outcomes and safety goals. Several different models will be presented to introduce examples of the various model(s) to the reader.

Figure 2-4 is a graphical representation of the relationship of the software development life cycle to the system/hardware development life cycle. Note that software life cycle graphic shown in Figure 2-4 portrays the DOD-STD-2167A software life cycle, which was replaced with MIL-STD-498, dated December 5, 1994. The minor changes made to the software life cycle by MIL-STD-498 are also shown. Notice also, that the model is representative of the "Waterfall", or "Grand Design" life cycle. While this model is still being used on numerous procurements, other models are more representative of the current software development schemes currently being followed, such as the "Spiral" and "Modified V" software development life cycles.

It is important to recognize that the software development life cycle does not correlate exactly with the hardware (system) development life cycle. It "lags" behind the hardware development at the beginning but finishes before the hardware development is completed. It is also important to realize that specific design reviews for hardware usually lag those required for software. The implications will be discussed in Section 4 of this Handbook.

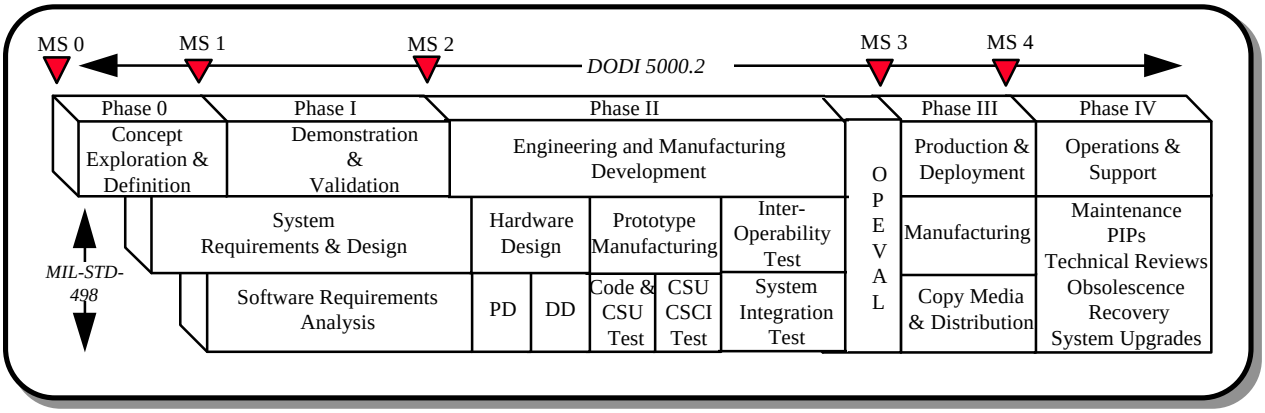
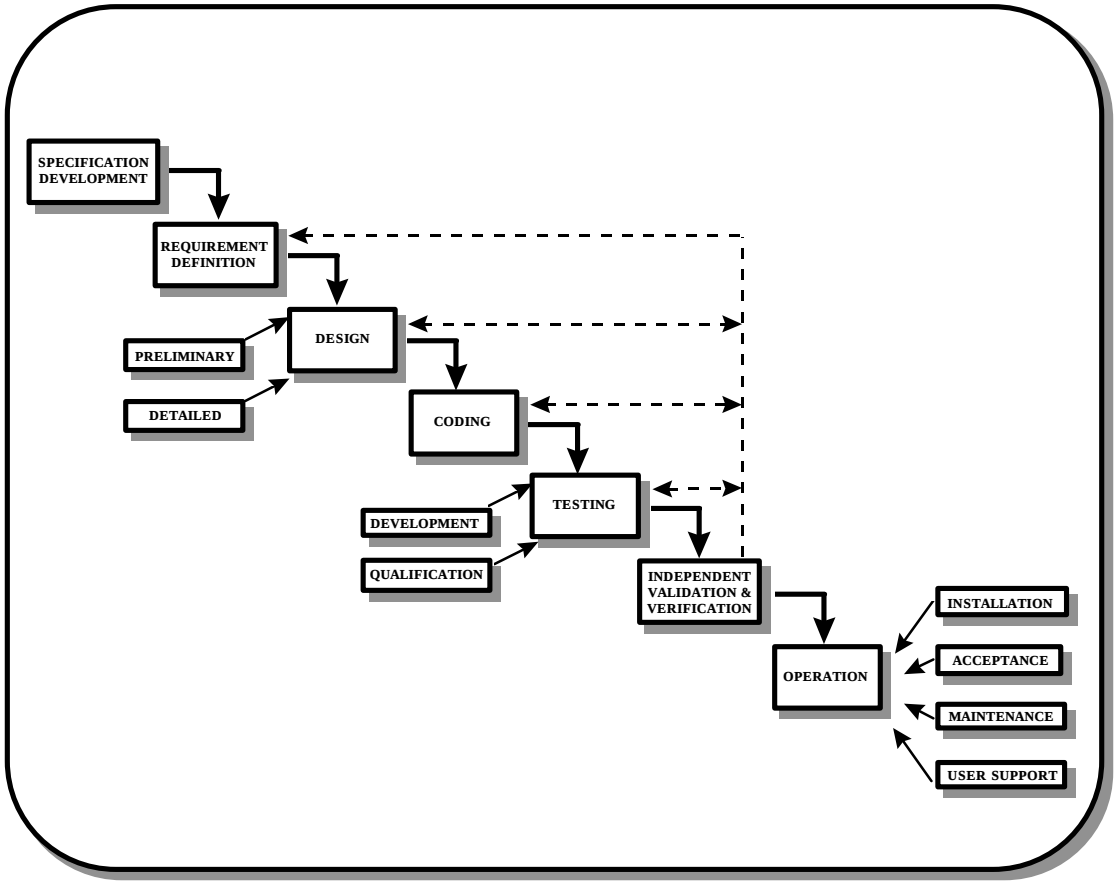


Figure 2-4: Relationship of Software to Hardware Development Life cycle

2.5.4.2.1 GRAND DESIGN, WATERFALL LIFE CYCLE MODEL[3]

The waterfall software acquisition and development life cycle model is the oldest in terms of use by software developers. This strategy usually uses DoD 2167A terminology and "...was conceived during the early 1970's as a remedy to the *code-and-fix* method of software development." Grand Design places emphasis on up-front documentation during early development phases, but does not support modern development practices such as prototyping and automatic code generation. "With each activity as a prerequisite for succeeding activities, this strategy is a risky choice for unprecedented systems because it inhibits flexibility." Another limitation to the model is that after a single pass through the model, the system is complete. Therefore, most integration problems are identified much too late in the development process to be corrected without significant cost and schedule impacts. In terms of software safety, interface issues must be identified and rectified as early as possible in the development life cycle to be adequately corrected and verified. Figure 2-5 is a representation of the grand design, or waterfall, life cycle model. The waterfall model is not recommended for large, software-intensive, systems. This is due to the limitations stated above, and the inability to effectively manage program risks, including safety risk during the software development process. The grand design does however provide a structured, disciplined, method for software development.



[3] Descriptions of the software acquisition life cycle models are either quoted or paraphrased from the *Guidelines for Successful Acquisition and Management of Software Intensive Systems,* STSC, September 1994, unless otherwise noted.

Figure 2-5: Waterfall, Grand Design Software Acquisition Model

2.5.4.2.2 MODIFIED V, LIFE CYCLE MODEL

The Modified V software acquisition life cycle model is another example of a defined method for software development. It is depicted in Figure 2-6. This model is heavily weighted in the ability to design, code, prototype, and test in increments of design maturity. "The left side of the figure identifies the specification, design, and coding activities for developing software. It also indicates when the test specification and test design activities can start. For example, the system/acceptance tests can be specified and designed as soon as software requirements are known. The integration tests can be specified and designed as soon as the software design structures are known. And, the unit tests can be specified and designed as soon as the code units are prepared."[4] The right side of the figure identifies when the evaluation activities occur that are involved with executing and testing the code at its various stages of evolution.

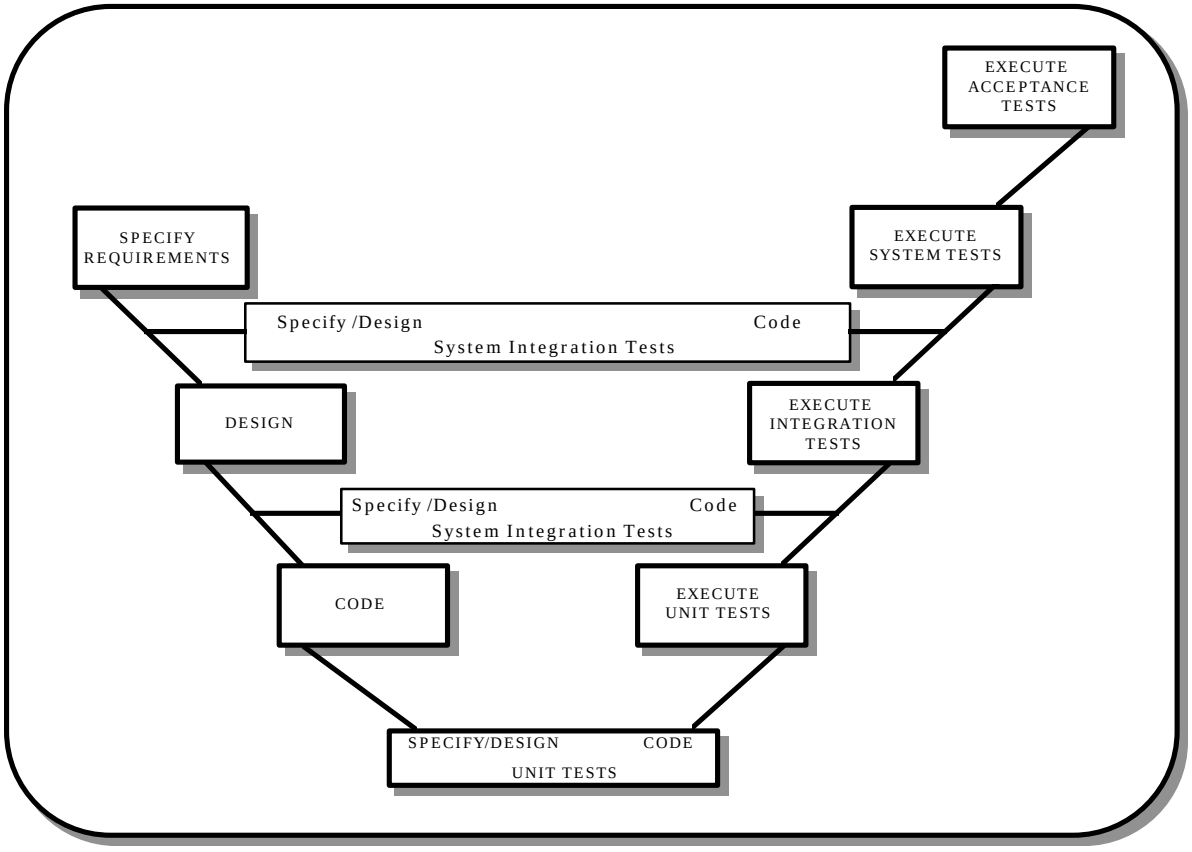


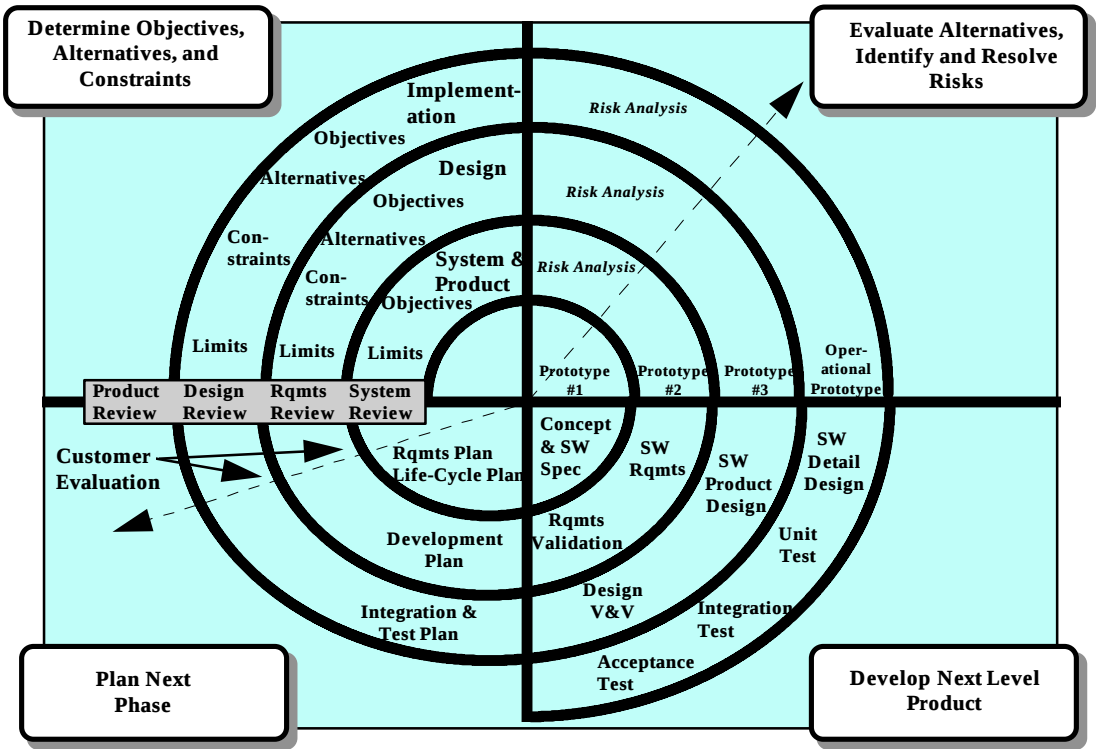
Figure 2-6: Modified "V" Acquisition Life Cycle Model

[4] *Software Test Technologies Report,* August 1994, Software Technology Support Center (STSC), Hill AFB, UT 84056

2.5.4.2.3 SPIRAL LIFE CYCLE MODEL

The spiral acquisition life cycle model provides a risk-reduction approach to the software development process. In the spiral model, Figure 2-7, the radial distance is a measure of effort expended, while the angular distance represents progress made. It combines features of the waterfall and the incremental prototype approaches to software development. "Spiral development emphasizes evaluation of alternatives and risk assessment. These are addressed more thoroughly than with other strategies. A review at the end of each phase ensures commitment to the next phase or identifies the need to rework a phase if necessary. The advantages of spiral development are its emphasis on procedures, such as risk analysis, and its adaptability to different development approaches. If spiral development is employed with demonstrations and baseline/configuration management, you can get continuous user buy-in and a disciplined process."[5]

Within the DoD, an ADA Spiral Model Environment is being considered for some procurements where the ADA language is being used. It provides an environment that combines a model and a tool environment, such that it offers the ability to have continual *touch-and-feel* of the software product (as opposed to paper reports and descriptions). This model represents a "demonstration-based" process that employs a top-down incremental approach which results in an early and continuous design and implementation validation. Advantages of this approach are that it is built from the top down, it supports partial implementation, the structure is automated, real and evolved, and that each level of development can be demonstrated. Each build and subsequent demonstration validates the process and the structure to the previous build.



[5] *Guidelines for Successful Acquisition and Management of Software Intensive Systems*, STSC, September 1994.

Figure 2-7: Spiral Acquisition Life Cycle Model

2.5.4.3 THE INTEGRATION OF HARDWARE AND SOFTWARE LIFE CYCLES

The life cycle process of system development is instituted so managers are not forced to make snap decisions. A structured life cycle, complete with controls, audits, reviews, and key decision points, provides a basis for sound decision making based on knowledge, experience, and training. It is a logical flow of events representing an orderly progression from a "user need" to final activation, deployment, and support.

The "systems approach" to software safety engineering must support a structured, well disciplined, and adequately documented system acquisition life cycle model that incorporates both the system development model and the software development model. Program plans (as described in Appendix C.7) must describe in detail how each engineering discipline will interface and perform within the development life cycle. It is recommended that you refer back to Figure 2-4 and review the integration of the hardware and software development life cycle models. Graphical representations of the life cycle model of choice for a given development activity must be provided during the planning processes. This activity will aid in the planning and implementation processes of software safety engineering. It will allow for the integration of safety-related requirements and guidelines into the design and code phases of software development. It will also assist in the timely identification of safety-specific test and verification requirements to prove that original design requirements have been implemented as they were intended. It further allows for the incorporation of safety inputs to the prototyping activities to demonstrate safety concepts.

2.5.5 A "TEAM" SOLUTION

The system safety engineer can not reduce the safety risk of systems software by himself. The software safety process must be an integrated team effort between engineering disciplines. Previously depicted in Figure 2-1, software, safety, and systems engineering are the pivotal players of the team. The management block is analogous to a "conductor" that provides the necessary motivation, direction, support, and resources for the team to perform as a well-orchestrated unit.

It is the intent of the authors of this Handbook to demonstrate that neither the software developers, nor safety engineers, can accomplish the necessary tasks, to the level of detailed required, by themselves. The Handbook will focus on the required tasks of the safety engineer, the software engineer, the software safety engineer, the system and design engineers, and the interfaces between each. Regardless of who executes the individual software safety tasks, each cognizant engineer must be intimately aware of the duties, responsibilities, and tasks required from each functional discipline. Each must also understand the time (in terms of life cycle schedule), place (in terms of required audits, meetings, reviews, etc.), and functional analysis tasks that must be produced and delivered at any point in the development process. Section 4 will expand on the team approach in detail as the planning, process tasks, products, and risk assessment tasks are presented. Figure 2-8 uses a puzzle analogy to demonstrate that the software safety approach must establish an integration between functions and among engineers.

Any piece of the puzzle that is missing from the picture will propagate into an unfinished or incomplete software safety work.

The elements contributing to a credible and successful software safety engineering program will include a defined and established system safety engineering process, a structured and disciplined software development process, an established hardware and software systems engineering process, an established hardware/software configuration control process, and an integrated software system safety team responsible for the identification, implementation, and verification of safety-specific requirements in the design and code of the software.

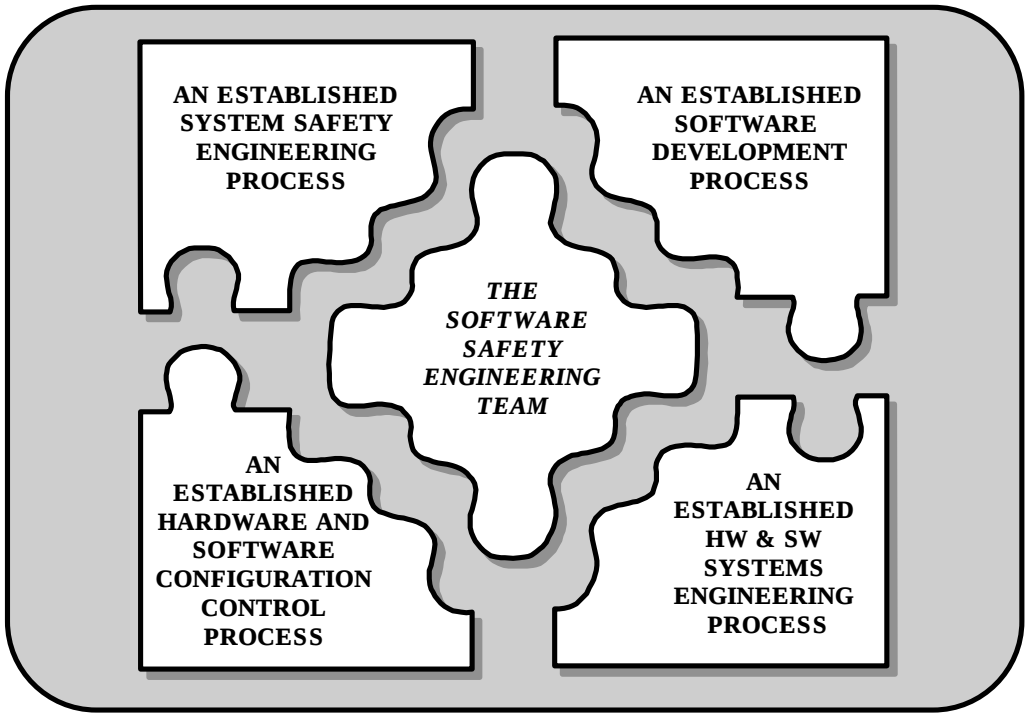


Figure 2-8: Integration of Engineering Personnel and Processes

2.6 HANDBOOK ORGANIZATION

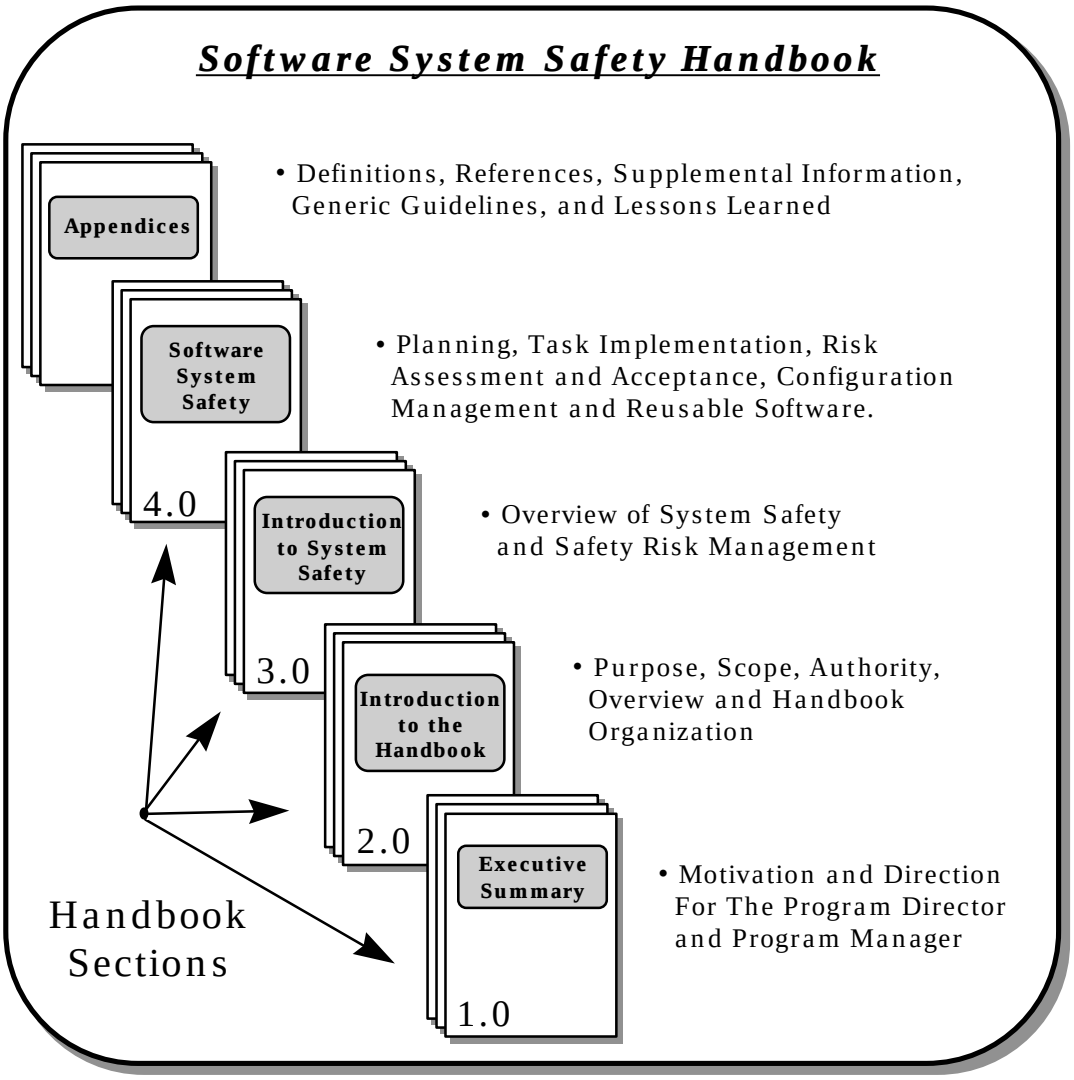


Figure 2-9: Handbook Layout

The Handbook is organized to provide the capability to review or extract subject information important to the reader. For example, the executive summary may be the only portion required by the executive officer, program director or program manager to determine whether a software safety program is required for their program. It is to be hoped that, the executive section will provide the necessary motivation, authority, and impetus for establishing a software safety program consistent with the nature of their development. Engineering and software managers, on the other hand, will need to read further into the document to obtain the managerial and technical process steps required for a software-intensive, safety-critical system development program. Safety program managers, safety engineers, software engineers, systems engineers, and software safety engineers will need to read even further into the document to gather the information necessary to develop, establish, implement, and manage an effective software system safety program. This includes the "how-to" details for conducting various analyses required to ensure that the system software will function within the system context to an acceptable level of safety

risk. Figure 2-9 graphically depicts the layout of the four handbook sections, their appendices, and provides a brief description of the contents of each.

As shown in Figure 2-9, Section 1.0 provides an executive overview of the handbook for the purpose of providing executive management a simplified overview of the subject of software safety, the requirement and authority for a SSS program, motivation and authority for the requirement, and their roles and responsibilities to the customer, the program, and to the design and development engineering disciplines. Section 2.0 provides an in-depth description of the purpose and scope of the Handbook, and the authority for the establishment of a software system safety program on DoD procurements and acquisition research and development activities. It also provides a description of the layout of the handbook as it applies to the acquisition life cycle of a system development. Section 3.0 provides an introduction to system safety engineering and management for those readers not familiar with the MIL-STD-882 methods and the approach for

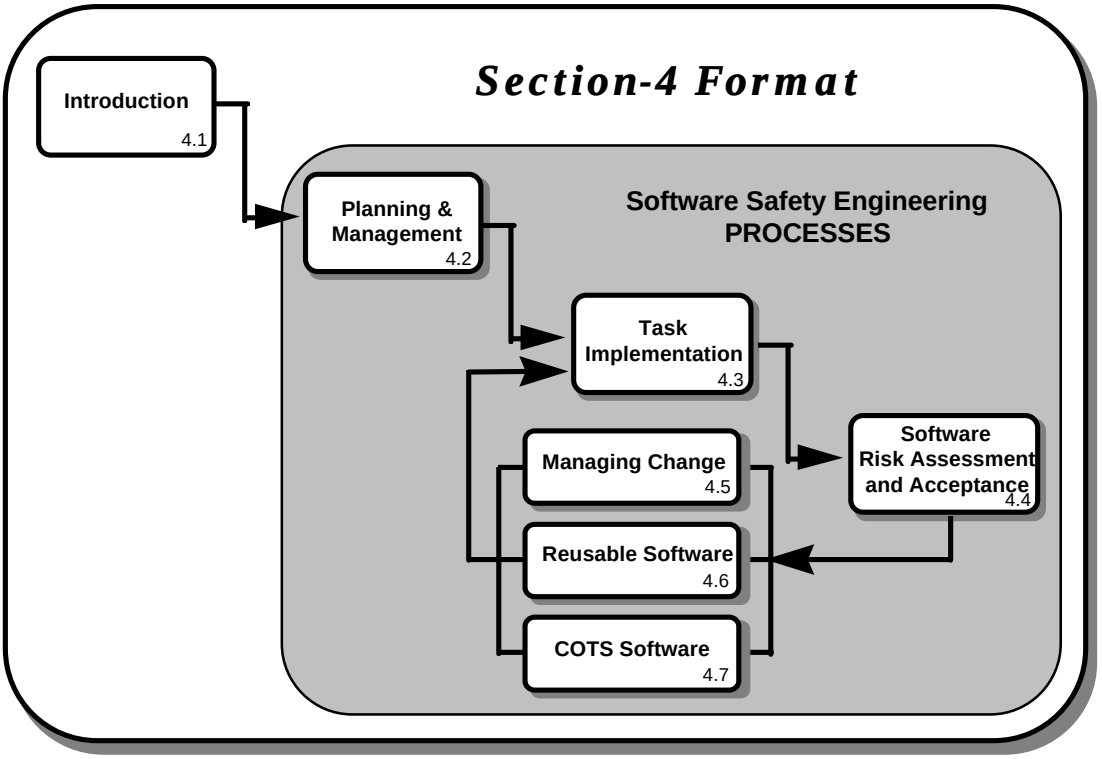


Figure 2-10: Section 4 Format

the establishment and implementation of a system safety program. It also provides an introduction to risk management and how safety risk is an integral part of the risk management function. Section 3.0 also provides an introduction to, and an overview of, the system acquisition, systems engineering, and software development process and guidance for the effective integration of these efforts in a comprehensive systems safety process. Section 4.0 provides the "how-to" of a baseline software safety program. The authors recognize that not all acquisitions and procurements are similar, nor do they possess the same problems, assumptions, and limitations in terms of technology, resources, development life cycles, and personalities. This section provides the basis for careful planning and forethought required to establish, tailor, and

implement a software system safety program. It is essential that the presented information is used as guidance for the practitioner, and not as a mindless "checklist" process.

Section 4.0, Software System Safety, is formatted logically (see Figure 2-10) to provide the reader with the steps required for planning, task implementation, and risk assessment and acceptance for a software system safety program. Section 4.0 also provides information regarding the management of configuration changes and issues pertaining to software reuse and commercial-off-the-shelf (COTS) software packages.

2.6.1 PLANNING

Section 4.0 begins with the planning required to establish a software system safety program. It discusses the program interfaces, contractual interfaces and obligations, safety resources, and program planning and plans. This particular section assists and guides the safety manager and engineer in the required steps of software safety program planning. Although there may be subject areas that are not required for each and every product procurement, each should be addressed and considered in the planning process. It is acceptable to determine that a specific activity or deliverable is not appropriate or necessary for *your* individual program.

2.6.2 TASK IMPLEMENTATION

This is the very heart of the handbook as applied to implementing a credible software safety program. It establishes a step-by-step baseline of "best practices" of today's approach in reducing the safety risk of software performing safety-critical and safety-significant functions within a system. A caution at this point is to **not** consider these process steps completely serial in nature. Although they are presented in a near serial format (for ease of reading and understanding), there are many activities that will require parallel processing and effort from the safety manager and engineer. Activities as complicated and as interface dependent as a software development within a systems acquisition process will seldom have required tasks line up where one task is complete before the next one begins. This is clearly demonstrated by the development of a software system safety program and milestone schedule (see Section 4.3.1)

This section of the handbook describes the tasks associated with contract and deliverable data development (including methods for tailoring), safety-critical function identification, preliminary and detailed hazard analysis, safety-specific requirements identification, implementation, test and verification, and residual risk analysis and acceptance. It also includes the participation in trade studies and design alternatives.

2.6.3 RISK ASSESSMENT AND ACCEPTANCE

The risk assessment and acceptance portion of Section 4.0 focuses on the tasks required to identify residual safety risk in the design, test, and operations of the system. It includes the evaluation and categorization of hazards remaining in the system and their impact to operations, maintenance and support functions. It also includes the graduated levels of programmatic sign-off for hazard and failure mode records of the subsystem, system, and operations & support hazard analyses. This section includes the tasks required to identify the hazards remaining in the system, assess their safety risk impact with their severity, probability or software control criticality, and determine the residual safety risk.

2.6.4 SUPPLEMENTARY APPENDICES

The handbook appendices include acronyms, definition of terms, handbook references, supplemental system safety information, generic safety requirements and guidelines, and lessons learned pertaining to the accomplishment of the software system safety tasks.

3. INTRODUCTION TO RISK MANAGEMENT AND SYSTEM SAFETY

3.1 INTRODUCTION

This Section should be read by software system safety team members who are not familiar with system safety. It should also be read by anyone who feels a need to become more familiar with the concept of Hazard Risk Index (HRI) and how hazards are rationally assessed, analyzed, correlated and tracked.

Section 3 will discuss:

- Risk and its application to the Software System Safety Program (SWSSP)
- Programmatic Risks
- Safety Risks

3.2 A DISCUSSION OF RISK

Everywhere we turn we are surrounded by a multitude of risks, some large and some so minimal that they can easily be overlooked, but all demanding to be recognized (i.e., assessed) and dealt with (i.e., managed). We view taking risks as foolhardy, irrational, and to be avoided. Risks imposed on us by others are generally considered to be entirely unacceptable. Everything we do involves risk. It is an unavoidable part of our everyday lives.

Realistically, some mishap risk must be accepted. How much is accepted, or not accepted, is the prerogative of management. That decision is affected by many inputs. As tradeoffs are being considered and the design progresses, it may become evident that some of the safety parameters are forcing higher program risk. From the program manager's perspective, a relaxation of one or more of the established parameters may appear to be advantageous when considering the broader perspective of cost and performance optimization. The program manager frequently will make a decision against the recommendation of his system safety manger. The system safety manager must recognize such management prerogatives. However, the prudent program manager must make the decision whether to fix the identified problem or formally document acceptance of the added risk. Of course, responsibility of personnel injury or loss of life changes the picture considerably. When the program manager decides to accept risk, the decision must be coordinated with all affected organizations and then documented so that in future years everyone will know and understand the elements of the decision and why it was made.

Accepting risk is a action of both risk assessment and risk management. Risk acceptance is not a simple matter as it may first appear. Several points must be kept in mind.

- Risk is a fundamental reality.
- Risk management is a process of tradeoffs.
- Quantifying risk doesn't ensure safety.

- Risk is a matter of perspective.

Risk Perspectives. When discussing risk, we must distinguish between three different standpoints:

- Standpoint of an INDIVIDUAL exposed to a hazard.
- Standpoint of society. Besides being interested in guaranteeing minimum individual risk for each of its members, society is concerned about the total risk to the general public.
- Standpoint of the INSTITUTION RESPONSIBLE FOR THE ACTIVITY. The institution responsible for an activity can be a private company or a government agency. From their point of view, it is essential to keep individual risks of employees or other persons and the collective risk at a minimum. An institution's concern is also to avoid catastrophic accidents.

The system safety effort is an optimizing process that varies in scope and scale over the lifetime of the system. System safety program balance is the result of the interplay between system safety and the three very familiar basic program elements: cost, performance, and schedule. Without an acute awareness of the system safety balance on the part of both program management and the system safety manager, they cannot discuss when, where, and how much they can afford to spend on system safety. We cannot afford mishaps which will prevent the achievement of primary mission goal, nor can we afford systems which cannot perform because of overstated safety goals.

Management's Risk Review. The system safety program examines the interrelationships of all components of a program and its systems with the objective of bringing mishap risk or risk reduction into the management review process for automatic consideration in total program perspective. It involves the preparation and implementation of system safety plans; the performance *of* system safety analyses on both system design and operations, and risk assessments in support of both management and system engineering activities. The system safety activity provides the manager with a means of identifying what the mishap risk is, where a mishap can be expected to occur, and what alternate designs are appropriate. Most important, it verifies implementation and effectiveness of hazard control. What is generally not recognized in the system safety community is that there are no safety problems in system design. There are only engineering and management problems, which if left unresolved, can result in a mishap. When a mishap occurs, then it is a safety problem. Identification and control of mishap risk is an engineering and management function. This is particularly true of software safety risk.

3.3 TYPES OF RISK

There are various models describing risk. The model in Figure 3-1 follows the system safety concept of risk reduction.

Total risk is the sum of identified and unidentified risks.

Identified risk is that risk which has been determined through various analysis techniques. The first task of system safety is to make identified risk as large a piece of the overall pie as practical. The time and costs of analyses efforts, the quality of the safety program, and the state of technology impact the amount of risk identified.

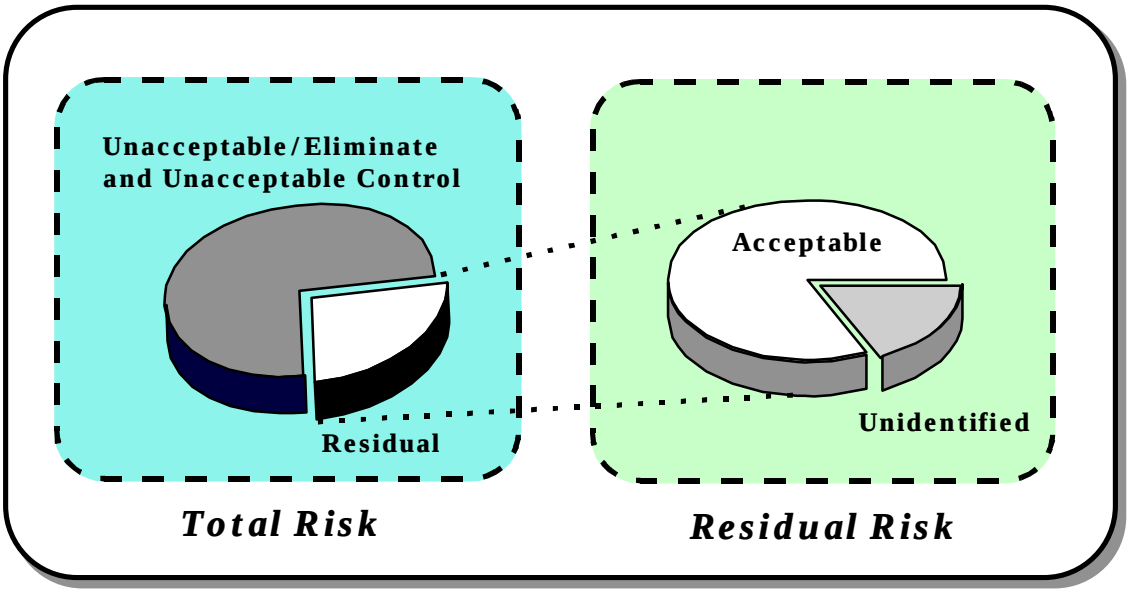


Figure 3-1: Types of Risk

Unacceptable risk is that risk which cannot be tolerated by the managing activity. It is a subset of identified risk which is either eliminated or controlled.

Residual risk is the risk left over after system safety efforts have been fully employed. It is sometimes erroneously thought of as being the same as acceptable risk. Residual risk is actually the sum of acceptable risk and unidentified risk. This is the total risk passed on to the user.

Acceptable risk is the part of identified risk which is allowed to persist without further engineering or management action. It is accepted by the managing activity. However, it is the user who is exposed to this risk.

Unidentified Risk is the risk that hasn't been determined. It's real. It's important. But It's not measurable. Some unidentified risk is subsequently determined when a mishap occurs. Some risk is never known.

3.4 AREAS OF PROGRAM RISK

Within the Department of Defense (DoD), risk is defined as a potential occurrence that is detrimental to either plans or programs. This risk is measured as the combined effect of the likelihood of the occurrence and a measured or assessed consequence given that occurrence [DSMC 1990]. The perceived risk to a program is usually different between program management, systems engineers, users, and safety. Because of this, the responsibility of defining program risk is usually assigned to a small group of individuals from various disciplines that can evaluate the program risks from a broad perspective of the total program concepts and issues to include business, cost, schedule, technical, and programmatic considerations. Although risk management responsibility is assigned to an individual group, the successful management of a programs risk is dependent on contributions and inputs of all individuals involved in the program management and engineering design functional activities.

In DoD, this risk management group is usually assigned to (or contained within) the systems engineering group. They are responsible for identifying, evaluating, measuring, and resolving risk within the program. This includes recognizing and understanding the warning signals that may indicate the program, or elements of the program, are off track. This risk management group must also understand the seriousness of the problems identified and then develop and implement plans to reduce the risk. A risk management assessment must be made early in the development life cycle and the risks must continually be reevaluated throughout the development life cycle. The members of the risk management group and the methods of risk identification and control should be documented in the programs' Risk Management Plan.

Risk Management[6] must consist of three activities:

Risk Planning - The process to force organized, purposeful thought to the subject of eliminating, minimizing, or containing the effects of undesirable consequences.

Risk Assessment - The process of examining a situation and identifying the areas of potential risk. The methods, techniques, or documentation most often used in risk assessment include:

- Systems engineering documents
- Operational Requirements Document (ORD)
- Operational Concepts Document (OCD)
- Life cycle cost analysis and models
- Schedule analysis and models
- Baseline cost estimates
- Requirements documents
- Lessons learned files and databases
- Trade studies and analyses
- Technical performance measurements and analyses
- Work Breakdown Structures (WBS)
- Project planning documents

Risk Analysis - The process of determining the probability of events and the potential consequences associated with those events relative to the program. The purpose of a risk analysis is to discover the cause, effects, and magnitude of the potential risk, and to develop and examine alternative actions which could reduce or eliminate these risks. Typical tools or models used in risk analysis include:

- Schedule Network Model
- Life Cycle Cost Model

[6] Selected descriptions and definitions regarding risk management are paraphrased from the Defense System Management College, Systems Engineering Management Guide, January 1990

- Quick Reaction Rate/Quantity Cost Impact Model
- System Modeling and Optimization

To further the discussion of program risk, short paragraphs are provided to help define schedule, budget, social-political, and technical risk. Although safety, by definition, is a part of technical risk, it can impact all areas of programmatic risk.

3.4.1 SCHEDULE RISK

The master system engineering and software development schedule for a program contains numerous areas of programmatic risk, such as schedules for new technology development, funding allocations, test site availability, critical personnel availability and rotation, etc. Each of these has the potential for delaying the development schedule and can induce unwarranted safety risk to the program. While these examples are by no means the only source of schedule risk, they are common to most programs. The risk manager must identify, analyze, and control risks to the program schedule by incorporating positive measures into the planning, scheduling, and coordinating activities for the purpose of minimizing their impact to the development program. To help accomplish these tasks the systems engineering function maintains the systems engineering master schedule (SEMS) and the systems engineering detailed schedule (SEDS). Maintaining these schedules helps to guide the interface between the customer and the developer; provides the cornerstone of the technical status and reporting process; and provides a disciplined interface between engineering disciplines and their respective system requirements. An example of the integration, documentation, tracking, and tracing of risk management issues is graphically depicted in Figure 3-2. Note that the SEMS and SEDS schedules, and the risk management effort is supported by a risk issue table and risk management database. These tools assist the risk manager in the identification, tracking, categorization, presentation, and resolution of managerial and technical risk.

Software developers for DoD customers or agencies are said to have maintained a perfect record to date. That is, they have never delivered a completed (meets all user/program requirements and specifications) software package on time yet. The inference is worthy of consideration. It implies that schedule risk is an important issue on a software development program. The schedule can become the driving factor forcing the delivery of immature and improperly tested critical software code to the customer. The risk manager, in concert with the safety manager, must ensure that the delivered product does not introduce safety risk to the user, system, maintainer, or the environment that is considered unacceptable. This is accomplished by the implementation of a software system safety program (and safety requirements) early in the software design process. The end result should produce a schedule risk reduction by decreasing the potential for re-design and re-code of software possessing safety deficiencies.

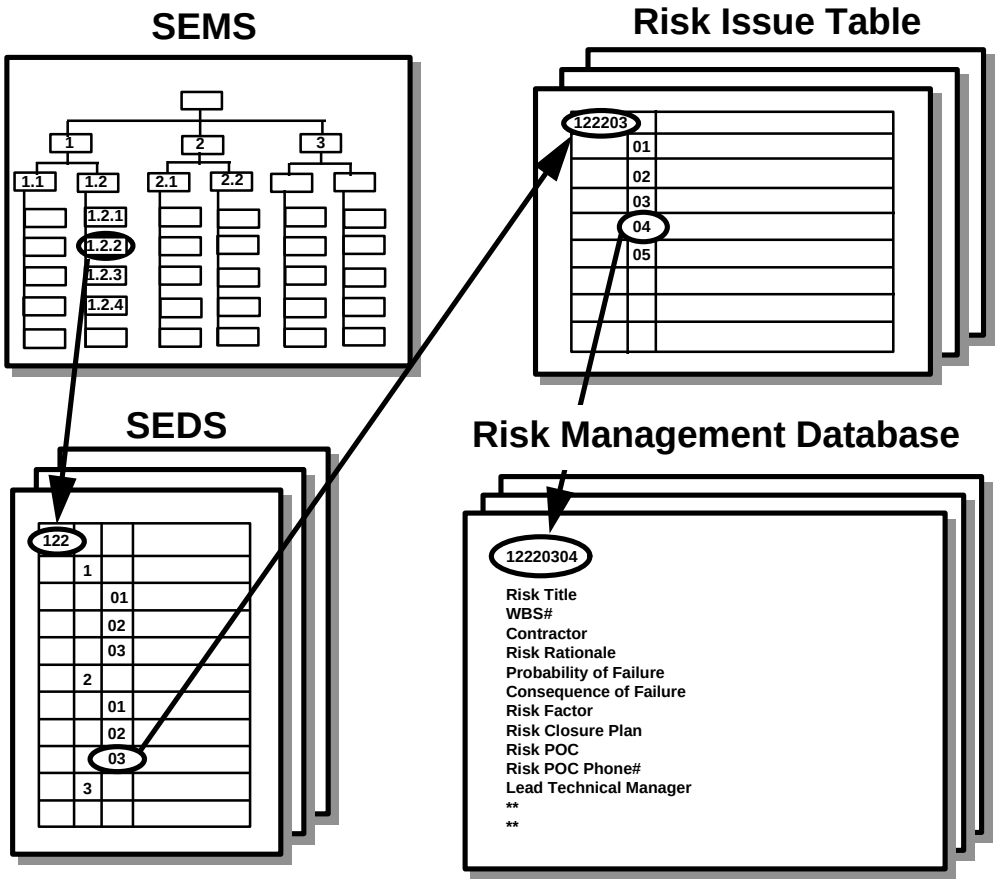


Figure 3-2: Systems Engineering, Risk Management Documentation

3.4.2 BUDGET RISK

Almost hand-in-hand with schedule risk comes budget risk. Although they can be mutually exclusive, that is seldom the case. The lack of monetary resources is always a potential risk in a development program. Within the DoD research, acquisition, and development agencies, the potential for budget cuts or congressional mandated program reductions seem always to be lurking around the next corner. Considering this potential, budgetary planning, cost scheduling, and program funding coordination become paramount to the risk management team. They must ensure that budgetary plans for current- and out-years are accurate and reasonable, and those potential limitations or contingencies to funding are identified, analyzed and incorporated into the program plans

In system safety terms, the development of safety-critical software requires significant program resources, highly skilled engineers, increased training requirements, software development tools, modeling and simulation, and facilities and testing resources. To ensure that this software meets functionality, safety and reliability goals, these activities become “drivers” for both program budget and schedule. Therefore, the safety manager must ensure that all safety-specific software development and test functions are prioritized in terms of safety risk potential to the program and to the operation of software after implementation. The prioritization of safety hazards and failure

modes, requirements, specifications, and test activities attributed to software help to facilitate and support the tasks performed by the risk management team. It will help them understand, prioritize and incorporate the activities necessary to minimize the safety risk potential for the program.

3.4.3 SOCIAL-POLITICAL RISK

This is a difficult subject to grasp from a risk management perspective. It is predicated more on public and political perceptions than basic truth and fact. Examples of this type of risk are often seen during the design, development, test, and fielding of a nuclear weapon systems in a geographical area that has a strong public, and possibly political, resistance. With an example like this in mind, several programmatic areas become important for discussion. First, program design, development, and test results have to be predicated on complete, technical fact. This will preclude any public perception of attempts to hide technical shortfalls or safety risk. Second, social and political perceptions can generate programmatic risk that must be considered by the risk managers. This includes the potential for budget cuts, schedule extensions, or program delays due to funding cuts as a result of public outcry and protest and its influence on politicians.

Safety plays a significant role in influencing the sensitivities of public or political personages toward a particular program. It must be a primary consideration in assessing risk management alternatives. If an accident (even a minor accident without injury) should occur during test, it could result in program cancellation.

3.4.4 TECHNICAL RISK

In system development and procurement, technical risk is where safety risk is most evident. Technical risk is the risk associated with the implementation of new technologies into the system being developed. This includes the hardware, software, human factors interface, and environmental safety engineering activities associated with the design, manufacture, fabrication, test, and deployment of the system. Technical risk results from poor identification and incorporation of system performance requirements to meet the intent of the user and system specifications. The inability to incorporate defined requirements into the design (lack of a technology base, lack of funds, lack of experience, etc.) increases the technical risk potential.

Systems engineers are usually tasked with activities associated with risk management. This is due to their assigned responsibility for technical risk within the design engineering function. The systems engineering function performs specification development, functional analysis, requirements allocation, trade studies, design optimization and effectiveness analysis, technical interface compatibility, logistic support analysis, program risk analysis, engineering integration and control, technical performance measurement, and documentation control.

The primary objective of risk management in terms of software safety is to understand that safety risk is a part of the technical risk of a program. All program risks must be identified, analyzed, and either eliminated or controlled. This includes safety risk, and thus, software (safety) risk.

3.5 SYSTEM SAFETY ENGINEERING

To understand the concept of system safety as it applies to software development, a basic introduction and description of system safety must be presented since software is a subset of the program system safety program activities. "System safety as we know it today began as a grass roots movement that was introduced in the 40's, gained momentum during the 50's, became established in the 60's, and formalized its place in the acquisition process in the 70's. The system safety concept was not the brain child of one man, but rather a call from the engineering and safety community to design and build safer equipment by applying lessons learned from our accident investigations."[7] It grew out of "conditions arising after WW II when its "parent" disciplines - systems engineering and systems analysis were developed to cope with new and complex engineering problems."[8] System Safety developed alongside, and in conjunction with, systems engineering and systems analysis... systems engineering being "the overall process of creating a complex human - machine system and systems analysis providing (1) the data for the decision - making aspects of that process and (2) an organized way to select among the latest alternative designs."[9] In the 1950's intense political pressure was focused on safety following several catastrophic mishaps. These included incidents where Atlas and Titan ICBM's exploded in their silos during operational testing. Investigation of the cause of these accidents revealed that a large percentage of the causal factors could be traced to deficiencies in design, operation and management that could have, and should have, been detected and corrected prior to the system being fielded. This recognition led to the development of system safety approaches to identify and control hazards in the systems design in order to minimize the likelihood and/or severity of first-time accidents.

As system safety analysis techniques and management methods evolved, they have been documented in various government and commercial standards. The first system safety specification was a document created by the Air Force in 1966; MIL-S-38130A. In June 1969, this standard was replaced by MIL-STD-882, and a systems safety program became mandatory for all DoD procured products and systems. Many of the later system safety requirements and standards in industry and other government agencies were developed based on MIL-STD-882, and remain so today. As both DoD and NASA began to use computers/software increasingly to perform critical system functions, concern about the safety aspects of these components began to emerge. The DoD initiated efforts to integrate software into system safety programs in the 1980's with the development of an extensive set of software safety tasks (300 series tasks) for incorporation into MIL-STD-882B (notice 1).

Although the identification of separate software safety tasks in MIL-STD-882B focused engineering attention on the hazard risks associated with the software components of a system and its critical effect on safety, it was perceived as "segregated" tasks to the overall system safety process and system safety engineers tried to push the responsibility for performing these tasks onto software engineers. Since software engineers had little understanding of the "system safety" process, and usually little knowledge of the overall system functional requirements, this was an

[7] Air Force System Safety Handbook, August 1992
[8] Leveson, Nancy G., *Safeware, System Safety and Computers*, 1995, Addison Wesley, page 129
[9] Ibid, page 143

unworkable process for dealing with software safety requirements. Therefore, the separate software safety tasks were not included in MIL-STD-882C as separate tasks, but were incorporated into the overall system-related safety tasks. In addition, software engineers were given a clear responsibility and a defined role in the SSS process in MIL-STD-498"

MIL-STD-882 defines system safety as:

"*The application of engineering and management principles, criteria, and techniques to optimize all aspects of safety within the constraints of operational effectiveness, time, and cost throughout all phases of the system life cycle*."

System safety program objectives can be further defined as:

- Safety, consistent with mission requirements, is designed into the system in a timely, cost-effective manner.
- Hazards associated with systems, subsystems, or equipment are identified, tracked, evaluated, and eliminated; or their associated risk is reduced to a level acceptable to the Managing Authority (MA) by evidence analysis throughout the entire life cycle of a system.
- Historical safety data, including lessons learned from other systems, are considered and used.
- Minimum risk consistent with user needs is sought in accepting and using new design technology, materials, production, tests, and techniques. Operational procedures must also be considered.
- Actions taken to eliminate hazards or reduce risk to a level acceptable to the MA are documented.
- Retrofit actions required to improve safety are minimized through the timely inclusion of safety design features during research, technology development for, and acquisition of, a system.
- Changes in design, configuration, or mission requirements are accomplished in a manner that maintains a risk level acceptable to the MA.
- Consideration is given early in the life cycle to safety and ease of disposal (including explosive ordnance disposal), and demilitarization of any hazardous materials associated with the system. Actions should be taken to minimize the use of hazardous materials and, therefore, minimize the risks and life cycle costs associated with their use.
- Significant safety data are documented as "lessons learned" and are submitted to data banks or as proposed changes to applicable design handbooks and specifications.
- Safety is maintained and assured after the incorporation and verification of engineering change proposals (ECP's) and other system-related changes.

With these definitions and objectives in mind, the system safety manager/engineer is the primary individual(s) responsible for the identification, tracking, elimination and/or control of hazards or

failure modes which exist in the design, development, test, and production of both hardware and software and their interfaces with the user, maintainer, and the operational environment. System safety engineering is a proven and credible function supporting the design and systems engineering process. The process steps for managing, planning, analyzing and coordinating system safety requirements are well established, and when implemented, successfully meet the above stated objectives.

These general system safety program requirements are as follows:

- Eliminate identified hazards or reduce associated risk through design, including material selection or substitution.
- Isolate hazardous substances, components, and operations from other activities, areas, personnel, and incompatible materials.
- Locate equipment so that access during operations, servicing, maintenance, repair, or adjustment minimizes personnel exposure to hazards.
- Minimize risk resulting from excessive environmental conditions (e.g., temperature, pressure, noise, toxicity, acceleration, and vibration).
- Design to minimize risk created by human error in the operation and support of the system.
- Consider alternate approaches to minimize risk from hazards that cannot be eliminated. Such approaches include interlocks, redundancy, fail-safe design, fire suppression, and protective clothing, equipment, devices, and procedures.
- Protect power sources, controls, and critical components of redundant subsystems by separation or shielding.
- When alternate design approaches cannot eliminate the hazard, provide warning and caution notes in assembly, operations, maintenance, and repair instructions, and distinctive markings on hazardous components and materials, equipment, and facilities to ensure personnel and equipment protection. These shall be standardized in accordance with MA requirements.
- Minimize severity of personnel injury or damage to equipment in the event of a mishap.
- Design software controlled or monitored functions to minimize initiation of hazardous events or mishaps.
- Review design criteria for inadequate or overly restrictive requirements regarding safety. Recommend new design criteria supported by study, analyses, or test data.

A good example of the need for, and the credibility of, a system safety engineering program is the Air Force aircraft mishap rate improvement since the establishment of the system safety program in the design, test, operations, support, and training processes. In the mid-1950's, the aircraft mishap rates were over 10 per 100,000 flight hours. Today this rate has been reduced to less than 1.25 per 100,000 flight hours.

Further information regarding the management and implementation of system safety engineering (and the analyses performed to support the goals and objectives of a system safety program) is available through numerous technical resources. It is not the intent of this Handbook to become another technical source book for the subject of system safety, but to address the implementation of software (system) safety within the discipline of system safety engineering. If specific system safety methods, techniques, or concepts remain unclear, please refer to the list of references in Appendix B for supplemental resources relating to the subject matter.

With the above information regarding system safety engineering (as a discipline) firmly in tow, a brief discussion must be presented as it applies to hazards and failure mode identification, categorization of safety risk in terms of probability and severity, and the methods of resolution. This concept must be firmly understood as the discussion evolves to the accomplishment of software safety tasks within the system safety engineering discipline.

3.6 SAFETY RISK MANAGEMENT

The process of system safety management and engineering is "deceptively simple"[10] although it entails a great deal of work. It is aimed at identifying system hazards and failure modes, determining their causes, assessing hazard severity and probability of occurrence, determining hazard control requirements, verifying their implementation, and identifying and quantifying any residual risk remaining prior to system deployment. Within an introduction of safety risk management, the concepts of hazard identification, hazard categorization and hazard risk reduction must be presented. Safety risk management focuses on the safety aspects of technical risk as it pertains to the conceptual system proposed for development. It attempts to identify and prioritize hazards that are most severe, and/or have the greatest probability of occurrence. The safety engineering process then identifies and implements safety risk elimination or reduction requirements for the design, development, test, and system activation phases of the development life cycle.

As the concept of safety risk management is further defined, keep in mind that the defined process is relatively simple and that the effort to control safety risk on a program will most likely be reduced if the process is followed.

3.6.1 INITIAL SAFETY RISK ASSESSMENT

The efforts of the system safety engineer is launched by the performance of the initial safety risk assessment of the system. In the case of most DoD procurements, this takes place with the accomplishment of the Preliminary Hazard List (PHL) and the Preliminary Hazard Analysis (PHA). This analysis is discussed in detail later in the handbook. The primary focus here, is the assessment and analysis of hazards and failure modes which are evident in the proposed system. This section of the Handbook will focus on the basic principles of system safety and hazard resolution. Specific discussions regarding how software influences or is related to hazards will be discussed in detail in Section 4.0.

[10] System Safety Analysis Handbook, A Resource Book For Safety Practitioners

3.6.1.1 HAZARD AND FAILURE MODE IDENTIFICATION

A hazard is defined as; *A condition that is prerequisite to a mishap.* These conditions, or hazards are identified by the system safety engineer. The initial hazard analysis, and the failure modes and effects analysis (FMEA) accomplished by the reliability engineer, provide the safety information required to perform the initial safety risk assessment of identified hazards. Without identified hazards and failure modes, very little can be accomplished to improve the overall safety of a system (remember this fact as software safety is introduced). Identified hazards and failure modes become the basis for the identification and implementation of safety requirements within the design of the system. Once the hazards are identified, they must be categorized in terms of safety risk.

3.6.1.2 HAZARD SEVERITY

The first step in classifying safety risk requires the establishment of hazard severity within the context of the system and user environments. This is typically done in two steps, first using the severity of damage and then applying the number of times the damage might occur. Table 3-1 provides an example of how severity can be qualified.

For Example Purposes Only

DESCRIPTION	*CATEGORY*	*SEVERITY OF HAZARD EFFECT*
Catastrophic	I	Death Or System Loss
Critical	II	Severe Injury, Occupational Illness, Major System Or Environmental Damage
Marginal	III	Minor Injury, Occupational Illness, Minor System Or Environmental Damage
Negligible	IV	Less Than Minor Injury, Illness, System Damage Or Environmental Damage

Table 3-1: Hazard Severity

Note that this example is typical to the MIL-STD-882 specified format. As you can see, the "severity of hazard effect" is qualitative and can be modified to meet the special needs of a program. There is an important aspect of the graphic to remember for any procurement. In order to assess safety severity, a benchmark to measure against is essential. The benchmark allows for the establishment of a qualitative baseline that can be communicated across programmatic and technical interfaces. It must be in a format language that make sense among individuals and between program interfaces.

3.6.1.3 HAZARD PROBABILITY

The second half of the equation for the determination of safety risk is the identification of the probability of occurrence. The probability that a hazard or failure mode may occur, given that it is not controlled, can be determined by numerous statistical techniques. These statistical probabilities are usually obtained from reliability analysis pertaining to hardware component failures acquired through qualification programs. The availability of component failure rates from reliability engineering is not always obtainable. This is especially true on advanced technology programs where component qualification programs do not exist and "one-of-a-kind" items are procured. Thus, the quantification of probability to a desired confidence level is not always possible for a specific hazard. When this occurs, alternative techniques of analysis are required for the qualification or quantification of hazard probability of hardware nodes. Examples of credible alternatives include sensitivity analysis, event tree diagrams (ETD), and fault tree analysis (FTA Fault Tree Analysis"). An example for the categorization of probability is provided in table 3-2 and is similar to the format recommended by MIL-STD-882.

For Example Purposes Only

DESCRIPTION	*LEVEL*	*PROGRAM DESCRIPTION*	*PROBABILITY*
Frequent	A	Will Occur	1 in 100
Probable	B	Likely To Occur	1 in 1000
Occasional	C	Unlikely To Occur, But Possible	1 in 10,000
Remote	D	Very Unlikely To Occur	1 in 100,000
Improbable	E	Assume It Will Not Occur	1 in 1,000,000

Table 3-2: Hazard Probability

As with the example provided for hazard severity, Table 3-2 can be modified to meet the specification requirements of the user and/or developer. A system engineering team (to include system safety engineering) may choose to shift the probability numbers an order of magnitude in either direction or to include, or reduce, the number of categories. All of the options are acceptable if the entire team is in agreement. This agreement must definitely include the customer's opinions and specification requirements. Also of importance when considering probability categories is the inclusion of individual units, entire populations, and time intervals (periods) which are appropriate for the system being analyzed.

3.6.1.4 HAZARD RISK INDEX (HRI) MATRIX

Hazard severity and hazard probability when integrated into a table format produces the Hazard Risk Index (HRI) matrix and the initial HRI risk categorization for hazards prior to control requirements are implemented. An example of a HRI matrix is provided in Table 3-3. This example was utilized on a research and development activity (X-30, National Aerospace Plane) where little component failure data was available. This matrix was separated into three levels of management acceptance. HRI's of 1-5 were considered hazards of unacceptable risk and required acquisition executive acceptance or rejection of safety risk. HRI's of 6-9 were considered moderate (to high) risk and required the program managers acceptance of residual risk. HRI's of 10-20 were considered lower risk hazards, and were put under the management of the lead design engineer and the safety manager.

Hazard Risk Index

For Example Purposes Only

Severity / Probability	I Catastrophic	II Critical	III Marginal	IV Negligible
(A) FREQUENT 1 IN 100	1	3	7	13
(B) PROBABLE 1 IN 1,000	2	5	9	16
(C) OCCASIONAL 1 IN 10,000	4	6	11	18
(D) REMOTE 1 IN 100,000	8	10	14	19
(E) IMPROBABLE 1 IN 1,000,000	12	15	17	20

Unacceptable Risk - Acquisition Executive Resolution Or Risk Acceptance

Marginal Risk - Design To Eliminate, Requires Program Manager Resolution Or Risk Acceptance

Minimum Risk - Design To Minimize, Requires Program Manager Resolution Or Risk Acceptance

Table 3-3: Hazard Risk Index

The true benefit of the HRI matrix is the ability and flexibility to prioritize hazards in terms of severity and probability. This prioritization of hazards allows the program manager, safety manager, and engineering manager the ability to also prioritize the expenditure and allocation of critical resources. Although it seems simplistic, a hazard with an HRI of eleven (11) should have fewer resources expended in its analysis, design, test, and verification than a hazard of four (4). Without the availability of the matrix, the allocation of resources becomes more arbitrary.

Another benefit of the Hazard Risk Index, is the accountability and responsibility of program and technical management to the system safety effort. The SSPP identifies and assigns specific levels of management authority with the appropriate levels of safety hazard severity and probability.

The HRI methodology holds program management and technical engineering accountable for the safety risk of the system during design, test and operation, and the residual risk upon delivery to the customer.

From the perspective of the safety analyst, the HRI is a tool that is used during the entire system safety effort throughout the product life cycle. Note however, that the HRI as a tool is somewhat confusing when applied to the evaluation of system hazards and failure mode influenced by software inputs or software information. Alternatives to the classical Hazard Risk Index are discussed in detail in Section 4.

3.6.2 SAFETY ORDER OF PRECEDENCE

The ability to adequately eliminate or control safety risk is predicated on the ability to accomplish the necessary tasks early in the design phases of the acquisition life cycle. For example, it is more cost effective and technologically efficient to eliminate a known hazard by changing the design (on paper), than retrofitting a fleet in operational use. Because of this, the system safety engineering methodology employees a safety order of precedence for hazard elimination or control. When incorporated, the design order of precedence further eliminates or reduces the severity and probability of hazard/failure mode initiation and propagation throughout the system. The following is extracted from MIL-STD-882C, paragraph 4.4.

- Design for Minimum Risk - From the first, design to eliminate hazards. If an identified hazard cannot be eliminated, reduce the associated risk to an acceptable level, as defined by the MA, through design selection.
- Incorporated Safety Devices - If identified hazards cannot be eliminated or their associated risk adequately reduced through design selection, that risk shall be reduced to a level acceptable to the MA through the use of fixed, automatic, or other protective safety design features or devices. Provisions shall be made for periodic functional checks of safety devices when applicable.
- Provide Warning Devices - When neither design nor safety devices can effectively eliminate identified hazards or adequately reduce associated risk, devices shall be used to detect the condition and to produce an adequate warning signal to alert personnel of the hazard. Warning signals and their application shall be designed to minimize the probability of incorrect personnel reaction to the signals and shall be standardized within like types of systems.
- Develop Procedures and Training - Where it is impractical to eliminate hazards through design selection or adequately reduce the associated risk with safety and warning devices, procedures and training shall be used. However, without a specific waiver from the MA, no warning, caution, or other form of written advisory shall be used as the only risk reduction method for Category I or II hazards. Procedures may include the use of personal protective equipment. Precautionary notations shall be standardized as specified by the MA. Tasks and activities judged to be safety-critical by the MA may require certification of personnel proficiency.

3.6.3 ELIMINATION OR RISK REDUCTION

The process of hazard and failure mode elimination or risk reduction is based on the design order of precedence. Once hazards and failure modes are identified by evidence analysis and categorized, specific (or functionally derived) safety requirements must be identified for incorporation into the design for the elimination or control of safety risk. Defined requirements can be applicable for any of the four categories of the defined order of safety precedence. For example, a specific hazard may have several design requirements identified for incorporation into the system design. However, to further minimize the safety risk of the hazard, supplemental requirements may be appropriate for safety devices, warning devices, and operator/maintainer procedures and training. In fact, most hazards have more than one design or risk reduction requirement unless the hazard is completely eliminated through the first (and only) design requirement. Figure 3-3 shows the process required to eliminate or control safety risk via the order of precedence described in paragraph 3.6.2.

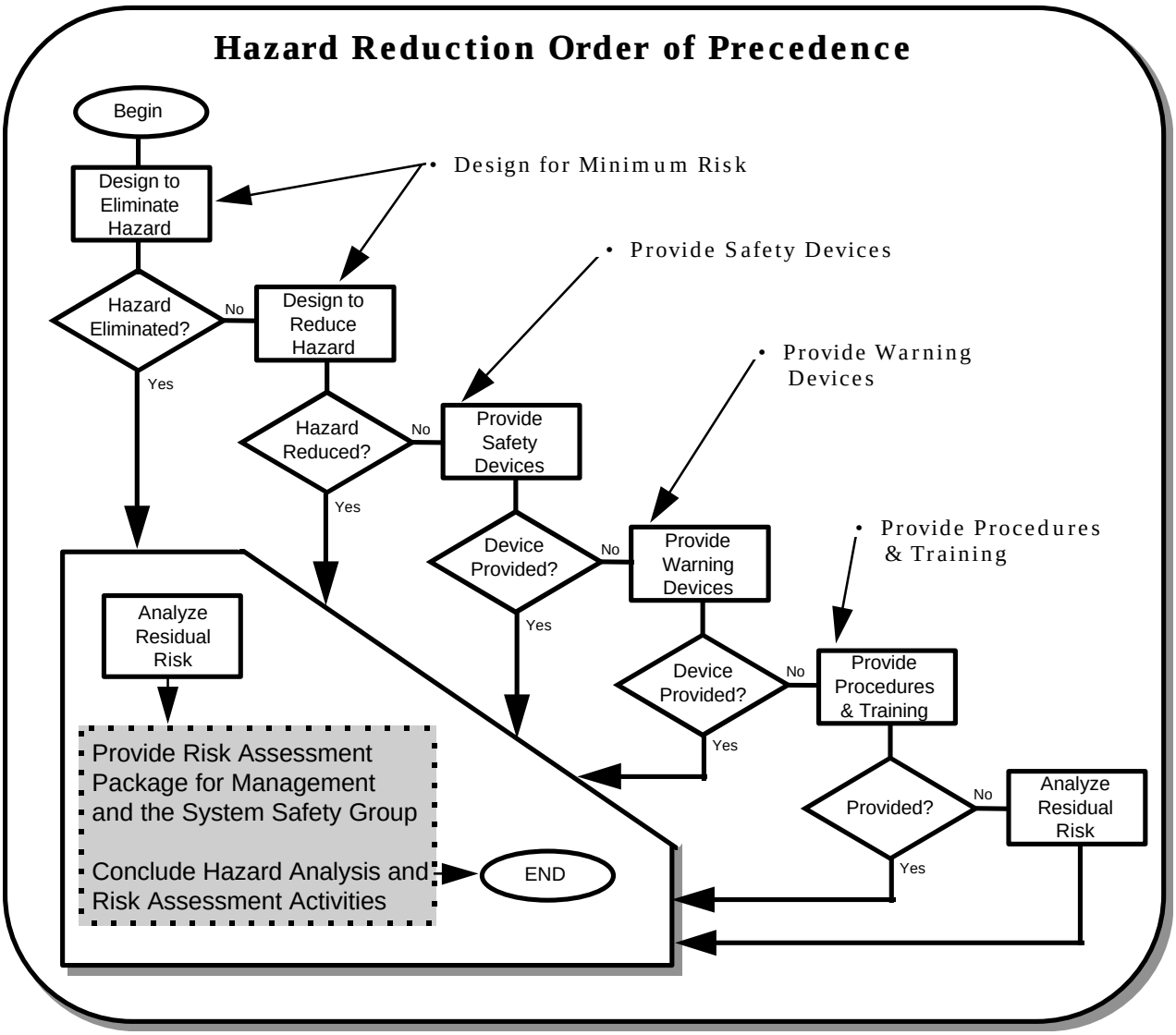


Figure 3-3: Hazard Reduction Order of Precedence

Identification of safety-specific requirements to the design and implementation portions of the system does not complete the safety task. The safety engineer must verify that the derived requirements have indeed been implemented as intended. Once hazard elimination and control requirements are identified and communicated to the appropriate design engineers, testing requirements must be identified for hazards which have been categorized as safety-critical. The categorization of safety risk in accordance with severity and probability must play a significant role in the depth of testing and requirements verification methods employed. Very low risk hazards do not require the same rigor of safety testing to verify the incorporation of requirements as compared to those associated with safety-critical hazards. Other verification methods may be more appropriate (i.e. designer sign-off on hazard record, as-built drawing review, inspection of manufactured components, etc.)

3.6.4 QUANTIFICATION OF RESIDUAL SAFETY RISK

After the requirements are implemented (to the extent possible), and appropriately verified in the design, the safety engineer must analyze each identified and documented hazard record to assess and analyze the residual risk that remains within the system during its operations and support activities. This is the same risk assessment process which was performed in the initial analysis described in Sections 3.6.1. The difference in the analysis is the amount of design and test data to support the risk reduction activities. After the incorporation of safety hazard elimination or reduction requirements, the hazard is once again assessed for severity, probability of occurrence, and an HRI determined. A hazard with an initial HRI of four (4), may have been reduced in safety risk to an HRI of eleven (11). However, since in this example, the hazard was not completely eliminated and only reduced, there remains a residual safety risk. Granted, it is not as severe or as probable as the original, but the hazard does exist. Remember, the initial HRI of a hazard is obtained prior to the incorporation or implementation of requirements to control or reduce the safety risk. The final HRI categorizes the hazard after the requirements have been implemented and verified by the developer. If hazards are not reduced sufficiently to meet the safety objectives and goals of the program, they must be reintroduced to engineering for further analyses and safety risk reduction.

In conjunction with the safety analysis, and the available engineering data and information available, residual safety risk of the system, subsystems, user, maintainer, and tester interfaces must be quantified. Hazard records with remaining residual risk must be correlated within subsystems, interfaces, and the total system for the purpose of calculating the remaining risk. This risk must be communicated in detail to the program manager (via the System Safety Working Group), the lead design engineers, the test manager, and the user. If residual risk in terms of safety is unacceptable to the program manager, further direction and resources must be provided to the engineering effort.

3.6.5 MANAGING AND ASSUMING RESIDUAL SAFETY RISK

Managing safety risk is another one of those “simple processes” which usually takes a great deal of time, effort, and resources to accomplish. Referring back to Table 3.3, Hazard Risk Index (HRI) Matrix, specific categories must be established on the matrix to identify the level of management accountability, responsibility, and risk acceptance. Using Table 3.3 as an example,

hazards with an HRI of between 1 through 5 are considered unacceptable. These hazards, if not reduced to a lower level below an HRI of 5, cannot be officially closed without the acquisitions executives' signature. This forces the accountability of assuming this particular risk to the appropriate level of management (the top manager). However, hazards from between an HRI of 6 through 20 can be officially closed by the program manager. This is to say that the program manager would be at the appropriate level of management to assume the safety risk to reduce the HRI to a lower category.

Recognize that Tables 3-1 through 3-3 are **for example purposes only**. They provide a graphical representation and depiction of how a program may be set up with three levels of program and technical management. It is ideal to have the program manager as the official sign-off for all residual safety risk to maintain safety accountability with that individual. Remember the program manager is responsible for the safety of a product or system at the time of test and deployment. The safety manager must establish an accountability system for the assumption of residual safety risk based upon user inputs, contractual obligations, and negotiations with the program manager.

4. SOFTWARE SAFETY ENGINEERING

4.1 INTRODUCTION

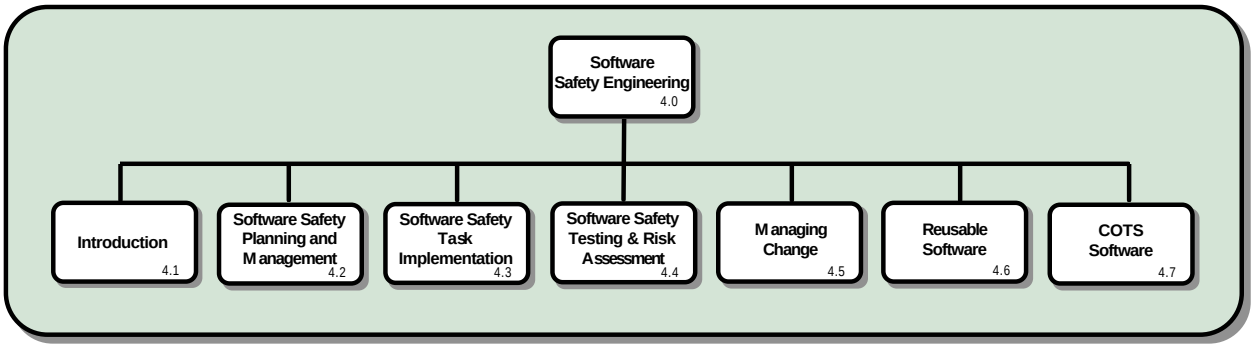


Figure 4-1: Section-4 Contents

This section of the handbook will introduce the managerial process and the technical methods and techniques inherent in the performance of software safety tasks within the context of a systems safety engineering and software development program. It will include detailed tasks and techniques for the performance of safety analysis, and for the traceability of software safety

considers the necessary steps in establishing a credible and cost-effective software safety program (Figure 4-1).

- Section 4 is applicable to all managerial and technical disciplines.
- Its primary purpose is to:
  a) Define a recommended software safety engineering process.
  b) Describe essential tasks to be accomplished by each professional discipline assigned to the software system safety team.
  c) Identify interface relationships between professional disciplines and the individual tasks assigned to the software system safety team.
  d) individual tasks.
  e) user requirements.

Section 4 is intended for the review and understanding of all systems engineers, system safety engineers, software safety engineers, and software development engineers. It is also appropriate for review by program managers interested in the technical aspects of the software system safety processes and the possible process improvement initiatives implemented by their systems engineers, software developers, design engineers, and programmers. This section will not only describe the essential tasks required by the system safety engineers, but also the required tasks

that must be accomplished by the software safety engineers, systems engineers, and the software developers. This includes the critical communication interfaces between each functional discipline. It also includes the identification, communication, and implementation of generic, or initial, software safety requirements and guidelines.

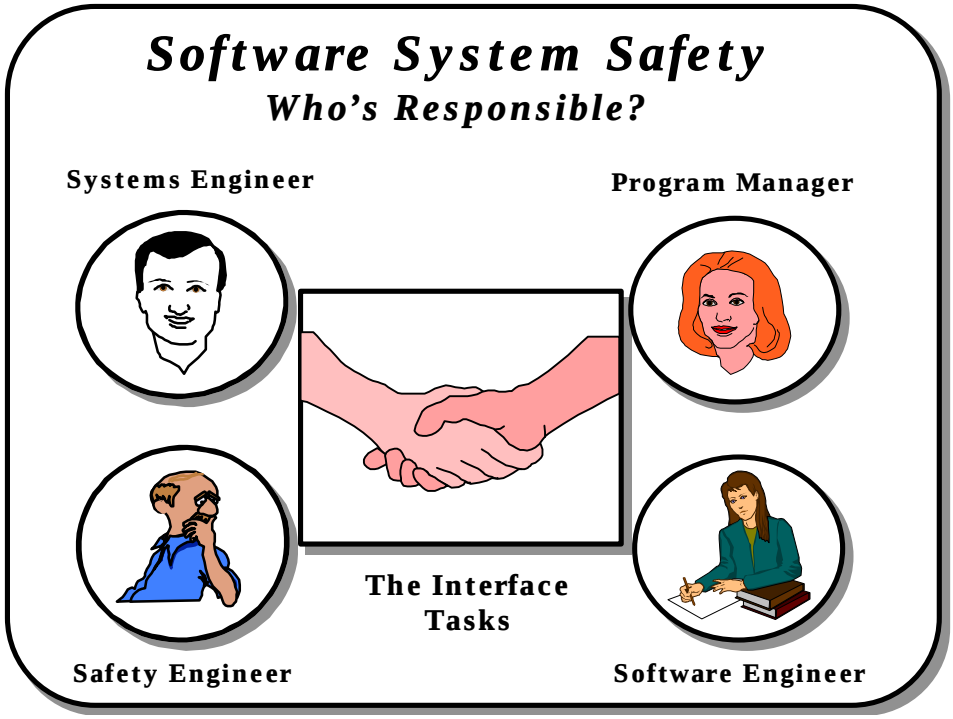


Figure 4-2: Who is Responsible for System Software Safety?

The accomplishment of a software safety management and engineering program requires careful forethought, adequate support from various other disciplines, and timely application of expertise across the entire software development life cycle. Strict attention to planning is required to integrate the developers resources, expertise, and experience with tasks to support contractual obligations established by the customer. This section focuses on the establishment of a software safety program within system safety engineering and the software development process. It establishes a baseline program that, when properly implemented, will ensure that both generic software safety requirements and requirements specifically derived from functional hazards analysis are identified, prioritized, categorized and traced through design, code, test, and acceptance.

A goal of this handbook section is to formally identify the software safety duties and responsibilities assigned to the safety engineer, the software safety engineer, the software engineer, and the managerial and technical interfaces of each through sound systems engineering methods (Figure 4-2). This handbook will identify and focus on the logical and practical relationships between the safety, design, and software disciplines. It will also provide the reader with the information necessary for the assignment of software safety responsibilities, and the identification of tasks attributed to system safety, software development, and hardware and digital systems engineering.

This handbook assumes a novice's understanding of software safety engineering within the context of system safety and software engineering. Note that many topics of discussion within this section are considered constructs within basic system safety engineering. This is due to the impossibility of discussing software safety outside the domain of system safety engineering and management, systems engineering, software development, and program management.

4.1.1 SECTION FOUR FORMAT

The software safety engineering section is formatted specifically to present both graphical and textual descriptions of the managerial and technical tasks that must be performed for a successful software safety engineering program. Each managerial process task, technical task, method, or technique will be formatted to provide the following:

- Graphical Representation of the Process Step or Technical Method
- Introductory and Supporting Text
- Prerequisite (Input) Requirements For Task Initiation
- Activities Required To Perform The Task (Including Inter-discipline Interfaces)
- Associated Subordinate Tasks
- Critical Task Interfaces
- And, A Description Of Required Task Output(s) And/Or Product(s)

This particular format helps to explain the task inputs, task activities, and task outputs for the successful accomplishment of activities to meet the goals and objectives of the software safety program. For those that desire additional information, Appendices A-H are provided to supplement the information in the main sections of this handbook. The appendices are intended to provide practitioners with supplemental information and credible examples for guidance purposes. The contents of the appendices include:

- Appendix A - Definition of Terms
- Appendix B - References
- Appendix C - Supplemental Information
- Appendix D - Generic Requirements and Guidelines
- Appendix E - Lessons Learned
- Appendix F - Detailed Process Chart Worksheets

4.1.2 PROCESS CHARTS

Each software safety engineering task possesses a supporting process chart. Each chart was developed for the purpose of providing the engineer with a detailed and complete "road map" for performing software safety engineering within the context of software design, code, and test activities. Figure 4-3 provides an example of the depth of information considered for each process task. The depth of information presented in this figure includes processes, methods,

documents, and deliverables associated with system safety engineering and management activities. However, for the purpose of this handbook, these process charts were "trimmed" to contain the information deemed essential for effective software safety program under the parent system safety program. The in-depth process chart worksheets are provided in Appendix F. for those interested in this detailed information.

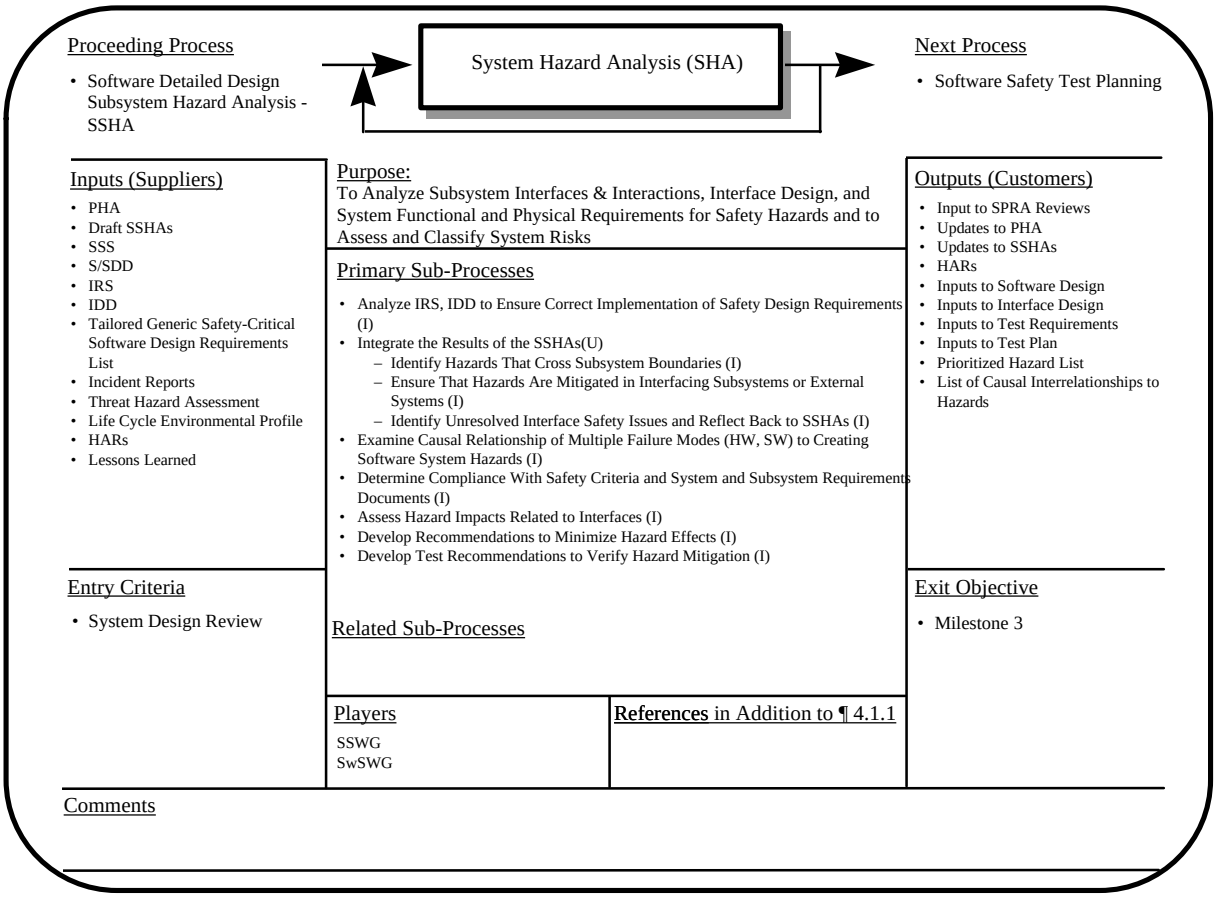


Figure 4-3: Initial Process Chart Example

Each process chart presented in this handbook will contain:

- Primary Task Description
- Task Inputs
- Task Outputs
- Primary Sub-Process Tasks
- Critical Interfaces

4.1.3 SOFTWARE SAFETY ENGINEERING PRODUCTS

The specific products to be produced by the accomplishment of the software safety engineering tasks are difficult to segregate from those developed within the context of the system safety program. It is likely, within individual programs, that supplemental software safety documents

and products will be produced to support the system safety effort. These may include supplemental analysis, data flow diagrams, functional flow analysis, and software requirement specifications. This handbook will identify and describe the documents and products that the software safety tasks will either influence or generate. Specific documents include:

- System Safety Program Plan
- Software Safety Program Plan
- Generic Software Safety Requirements List
- Safety-Critical Functions List
- Preliminary Hazard List
- Preliminary Hazard Analysis
- Sub-System Hazard Analysis
- Safety Requirements Criteria Analysis
- System Hazard Analysis
- Safety Assessment Report

4.2 SOFTWARE SAFETY PLANNING MANAGEMENT

The software safety program must parallel both the system safety program and the software development program milestones. The software safety analyses must provide the necessary input to software development milestones, such that, safety design requirements, implementation recommendations or changes can be incorporated into the software with minimal impact.

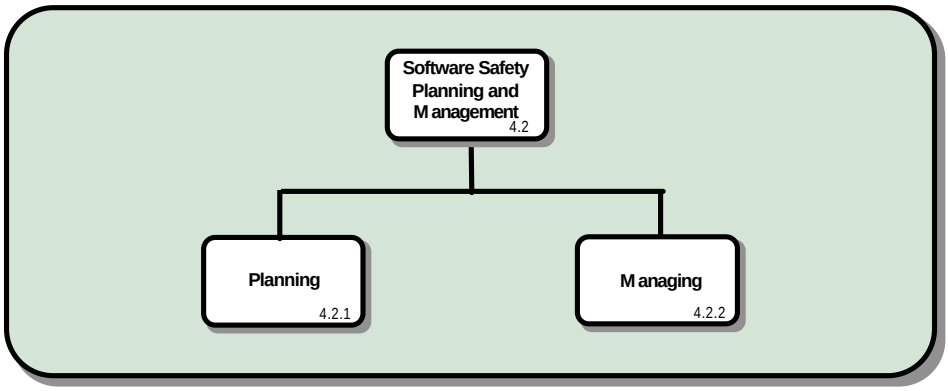


Figure 4-4: Software Safety Planning

The safety program planning precedes all other phases of the software systems safety program. It is perhaps the single most important step in the overall safety program. Inadequately specified safety requirements in the contractual documents generally leads to program schedule and cost impacts later when safety issues arise and the appropriate systems safety program has not been established. The software aspects of systems safety tends to be more problematic in this area since the risk associated with the software is often ignored or not well understood until late in the

system design. Establishing the safety program and performing the necessary safety analyses at later points in the program results in delays, cost increases, and a less effective safety program.

The history of software-related safety issues, as derived from lessons learned, establishes the requirement for a practical, logical, and disciplined approach for reducing the safety risk of software performing safety-critical functions within a system. This managerial and technical discipline must be established "up front" on the program and be included in the planning activities which both describes and documents the breadth and depth of the program. Detailed planning ensures that critical program interfaces and support is identified and formal lines of communication is established between disciplines and among engineering functions. The potential for program success is increased through sound planning activities which identify and formalize the managerial and technical interfaces of the program. Figure 4-4 demonstrates that the subject will be presented in terms of both planning and managing. To minimize the depth of the material presented, supporting and supplemental text is provided in Appendix C.

This section is applicable to all members of the software systems safety team. It assumes minimal understanding and experience with safety engineering programs. The topics that will be discussed include:

- Identification of managerial and technical program interfaces required by the software systems safety team.
- Definition of the user and developer contractual relationships to ensure the software systems safety team implements the tasks, and produces the products, required to meet contractual requirements.
- Identification of programmatic and technical meetings and reviews normally supported by the software system safety team.
- Identification and allocation of critical resources to establish a software systems safety team and conduct a software safety program.
- Definition of planning requirements for the execution of an effective program.

The planning for an effective system safety and software safety program requires extensive forethought from both the supplier and the customer. Although they both may have a perfect system safety program envisioned, there are subtle differences associated with the identification, preparation, and execution of a successful safety program from these two perspectives. The contents of Figures 4-5 and 4-6 represent the primary differences between agencies which must be understood before considering the software safety planning and coordinating activities.

4.2.1 PLANNING

Comprehensive planning for the software safety program requires an initial assessment of the degree of software involvement in the system design and the associated hazards. Unfortunately, this is difficult since little is usually known about the system other than operational requirements during the early planning stages. Therefore, the safety statement of work must be designed to encompass all possible designs. This generally results in a fairly generic statement of work that requires later tailoring of a software systems safety program to the system design.

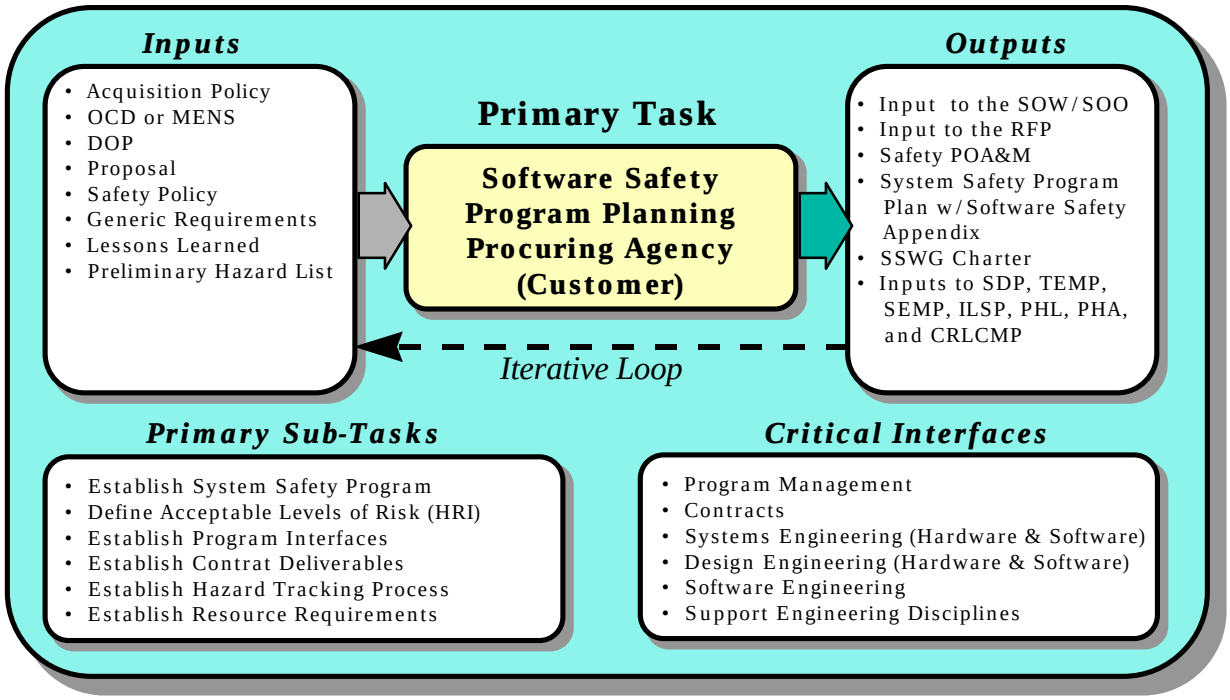


Figure 4-5: Software Safety Planning by the Procuring agency

Figure 4-5 represents the basic inputs, outputs, tasks, and critical interfaces associated with the planning requirements associated with the Procuring Agency (PA). Frustrations experienced by safety engineers executing the system safety tasks can usually be traced back to the lack of adequate planning by the customer. The direct result is normally an under-budget, under-staffed, safety effort that does not focus on the most critical aspects of the system being considered. This usually can be traced back to the customer not assuring that the Request for Proposal (RFP), Statement of Work (SOW), and the contract contain the correct language, terminology, and/or tasks to implement a safety program with the required or necessary resources. The ultimate success of any safety program is linked to this planning function by the customer.

For the procuring agency, software safety program planning begins as soon as the system need is identified. Points of contact within the PA are identified, interfaces between various engineering disciplines, administrative support organizations, program management, and the contracting group are defined, and Integrated Product Teams are established to begin development of the necessary requirements and specifications documents. In the context of acquisition reform, invoking military standards and specifications for DoD procurements is not permitted. Therefore, the necessary language to ensure that the system under development will meet the PA's safety goals and objectives must be incorporated into any contractual statements of work and specifications.

PA safety program planning continues through contract award and may require periodic updating during initial system development and as the development proceeds through various development phases. However, management of the overall system safety program continues through system delivery and acceptance and throughout the system's life cycle. Thus the PA must make provisions for safety program planning and management for any upgrades, product improvements, maintenance, and other follow-on efforts on the system. The major milestones affecting the PA's safety and software safety program planning include release of contract requests for proposals or

quotes, proposal evaluation, major program milestones, system acceptance testing and evaluation, production contract award, initial operational capability (release to the users), and system upgrades or product improvements.

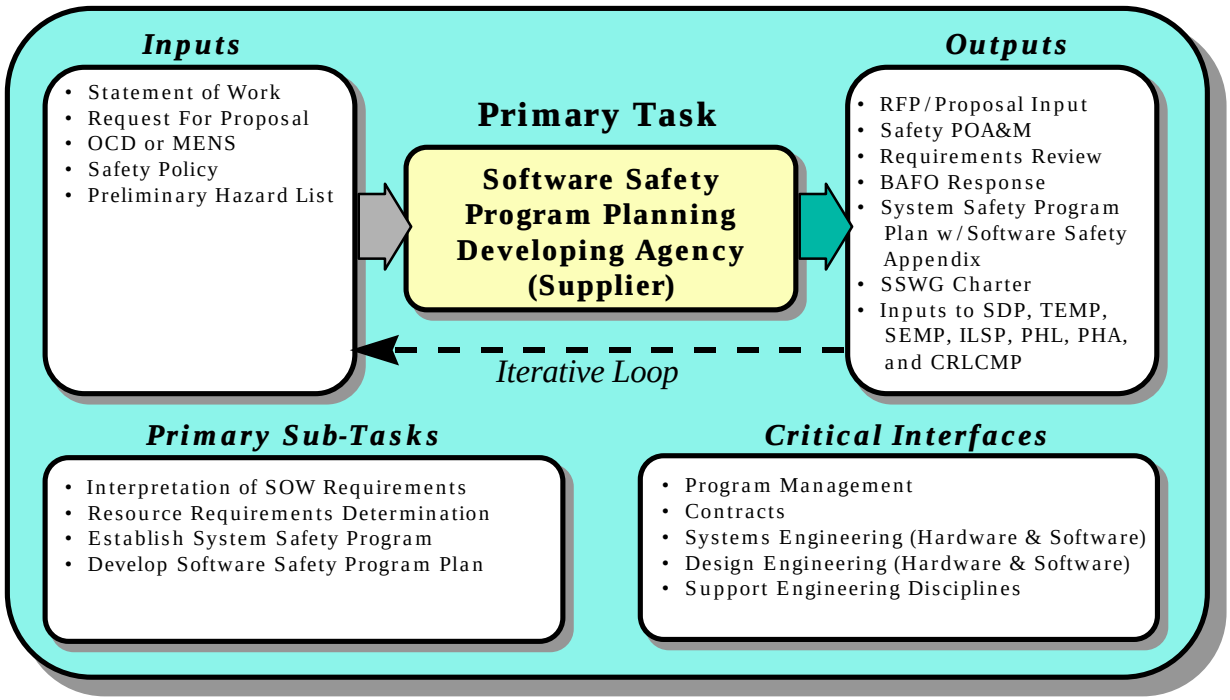


Figure 4-6: Software Safety Planning by the Developing Agency

Although software safety program planning begins after receipt of a contract RFP, or quotes, the Developing Agency (DA) can significantly enhance their ability to establish an effective program by prior planning (see Figure 4-6). However, acquisition reform now requires that the Government take more of a "hands-off" approach to system development. Although this means less interference by the procuring agency, fewer specifications and standards to comply with, and less stringent program requirements, it also requires the developing agency to warrant the system making the agency liable for any mishaps that occur with the system, even after acceptance by the procuring agency. Often, contract language is non-specific and doesn't provide detailed requirements, especially with respect to safety requirements and safety program requirements for the system. Therefore, it is the developing agency's responsibility to define a comprehensive system safety program that will ensure that the delivered system provides an acceptably low level of risk to the customer. At the same time, the developing agency must remain competitive and reduce safety program costs to the lowest practical level consistent with ensuring delivery of a system with the lowest risk practical.

Developing agency software safety planning continues after contract award and will require periodic updating during system development as it proceeds through various development phases. These updates should be in concert with the PA's software safety plans. However, management of the overall system and software systems safety programs continues from contract award through system delivery and acceptance. If the contract requires that the developing agency perform routine maintenance or system upgrades, the software safety program management may continue throughout the system's life cycle. Thus, the developing agency must make provisions

for safety program planning and management for any upgrades, product improvements, maintenance and other follow-on efforts on the system. The major milestones affecting the developing agency's safety and software safety program planning include receipt of contract requests for proposals or quotes, contract award, major program milestones, system acceptance testing and evaluation, production contract award, release to the customer, system upgrades, and product improvements.

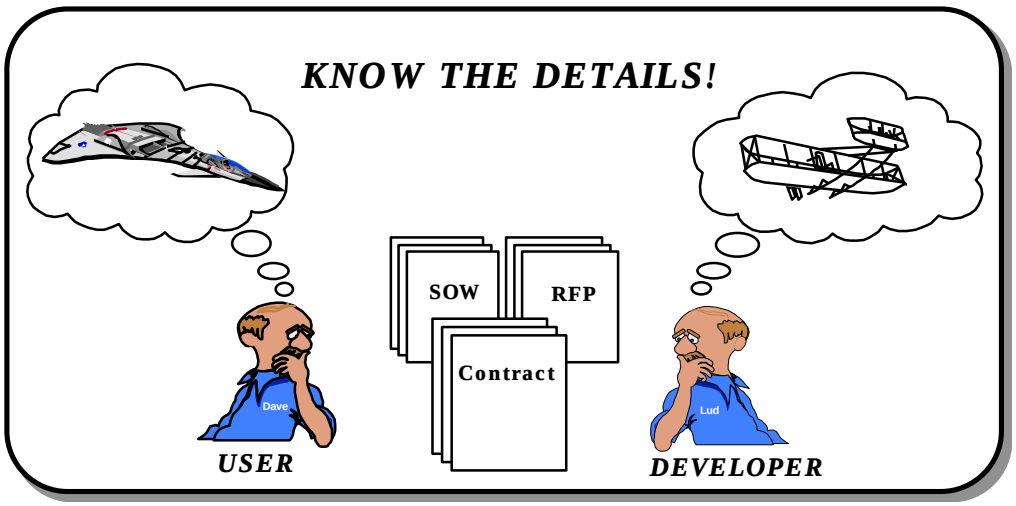


Figure 4-7: Planning The Safety Criteria is Important

While the objectives of software safety planning may be similar by the procuring agency and the developing agency, the planning and coordinating required to meet these objectives may come from slightly different angles (in terms of specific tasks and their implementation), but they must be in concert (see Figure 4-7). Regardless, both agencies must work together to meet the safety objectives of the program. In terms of planning, this includes the:

- Establishment of a system safety program
- Definition of acceptable levels of safety risk
- Definition of critical program, management, and engineering interfaces
- Definition of contract deliverables
- Development of a Software Hazard Criticality Matrix

4.2.1.1 ESTABLISH THE SYSTEM SAFETY PROGRAM

The PA safety program should be established as early as practical in the development of the system. A Principal For Safety (PFS) should be identified early who serves as the single point of contact for all safety related matters on the system. This individual will interface with safety review authorities, the developing agency safety team, program management, and other groups as required to ensure that the safety program is effective and efficient. The PFS will also establish and chair the Software Systems Safety Working Group (SwSWG) or Software System Safety (SSS) Team. For large system developments where software is likely to be a major portion of the development, a safety engineer for software may also be identified who reports directly to the overall system PFS. The size of the safety organization will depend on the complexity of the

system under development, and the inherent safety risks coupled with the degree of customer/supplier interaction and the other engineering and program disciplines. If the development approach is to be a team effort with a high degree of interaction between the organizations, as generally directed by the DOD acquisition instructions, additional personnel may be required to provide adequate support.

The software safety program must be specified for programs where software performs or influences safety-critical functions of the system. This program must be established in accordance with contractual requirements, managerial and technical interfaces and agreements, and the results of all planning activities discussed in previous sections of this handbook. Proper and detailed planning will increase the probability of program success. The tasks and activities associated with the establishment of the system safety program is applicable to both the supplier and the customer.

The establishment of the software safety program must be predicated on the goals and objectives of the system safety and the software development disciplines of the proposed program. This program must be focused on the identification and tracking (from design, code, and test) of both generic requirements and guidelines, and those requirements derived from system-specific, functional hazards analyses. It must be stated here, that common deficiencies of software safety programs are usually based upon the lack of a team approach in addressing both the generic and the functional software safety requirements of a system. The software development community has a tendency to focus on only the generic requirements while the system safety community may primarily focus on the functional requirements derived through hazards analysis. A sound software safety program will trace both sets of software safety requirements through test and requirements verification activities. The ability to identify (in total) all applicable safety requirements is considered essential for *any* given program and must be adequately addressed.

4.2.1.2 DEFINING ACCEPTABLE LEVELS OF RISK

One of the key elements in safety program planning is the identification of the acceptable level of risk for the system. This process requires both the identification of a Hazard Risk Index and a statement of the goal of the safety program for the system. The former establishes a standardized means with which to group hazards by risk (e.g., unacceptable, undesirable, etc.) while the latter provides a statement of the expected safety quality of the system.

FREQUENCY OF OCCURRENCE	HAZARD CATEGORIES			
	I CATASTROPHIC	II CRITICAL	III MARGINAL	IV NEGLIGIBLE
A - FREQUENT	0	0	0	0
B - PROBABLE	4	1	0	0
C - OCCASIONAL	5	16	0	0
D - REMOTE	24	25	3	0
E - IMPROBABLE	1	1	1	0

Legend:	
IA, IB, IC, IIA, IIB, IIIA	**UNACCEPTABLE**, condition must be resolved. Design action is required to eliminate or control hazard.
ID. IIC. IID. IIIB. IIIC	**UNDESIRABLE**, Program Manager decision is required. Hazard must be controlled or hazard probability reduced.
IE, IIE, IIID, IIIE, IVA, IVB	**ALLOWABLE**, with Program Manager review. Hazard control desirable if cost effective.
IVC, IVD, IVE	**ACCEPTABLE** without review. Normally not cost effective to control. Hazard is either negligible or can be assumed will not occur.

Figure 4-8: Risk Acceptance Matrix Example

The HRI (described in Section 3.6.1.4) is based on the criteria described in MIL-STD-882C. This example may be used for guidance, or a alternate hazard risk index may be proposed. The given HRI methodology used by a program must possess the capability to graphically delineate the boundaries between acceptable, allowable, undesirable, and unacceptable risk. Figure 4-8 provides a graphical representation of a risk acceptance matrix. In this example, the hazard record database contains ten (10) hazards which currently remain in the unacceptable categories (categories IA, IB, IC, IIA, IIB, and IIIA) of safety risk. This example explicitly states that the hazards represented in the unacceptable range must be resolved.

The ability to categorize specific hazards into the matrix above is based upon the ability of the safety engineer to assess the hazards' severity and likelihood of occurrence. Historically, the traditional HRI matrix (as described in Section 3.6.1.4), did not include the influence of the software on the hazard occurrence. The ability to assess the software's influence is based upon sound causal factor analyses and the software control capabilities described in Section 4.2.1.5.

4.2.1.3 PROGRAM INTERFACES

System Safety Engineering is responsible for the coordination, initiation and implementation of the software safety engineering program. This responsibility must not be delegated to any other engineering discipline within the development team, however, specific tasks must be assigned to the engineers with the appropriate expertise. Historically, system safety engineering has identified, eliminated, or reduced the safety risk of hazards associated with complex systems. Now, as software becomes a paramount dimension of the system, software safety engineering

must establish and perform the required tasks and establish the technical interfaces required to fulfill the goals and objectives of the system safety (and software safety) program. However, this cannot be accomplished independently without the inter-communication and support from other managerial and technical functions. Within the DoD acquisition and product development agencies, integrated product teams (IPT's) have been established to ensure the success of the design, manufacture, fabrication, test, and deployment of weapons systems. These IPT's formally establish the accountability and responsibility between functions and among team members. This accountability and responsibility is both from the top down (management to engineer), and from the bottom up (from the engineer to management).

The establishment of a credible software system safety activity within an organization requires this same rigor in the definition of team members, program interfaces, and lines of communication. The establishment of formal and defined interfaces allows program and engineering managers to assign required expertise for the performance of the identified tasks of the software safety engineering process. Figure 4-9 shows the common interfaces to adequately support a software system safety program. It includes management interfaces, technical interfaces, and contractual interfaces.

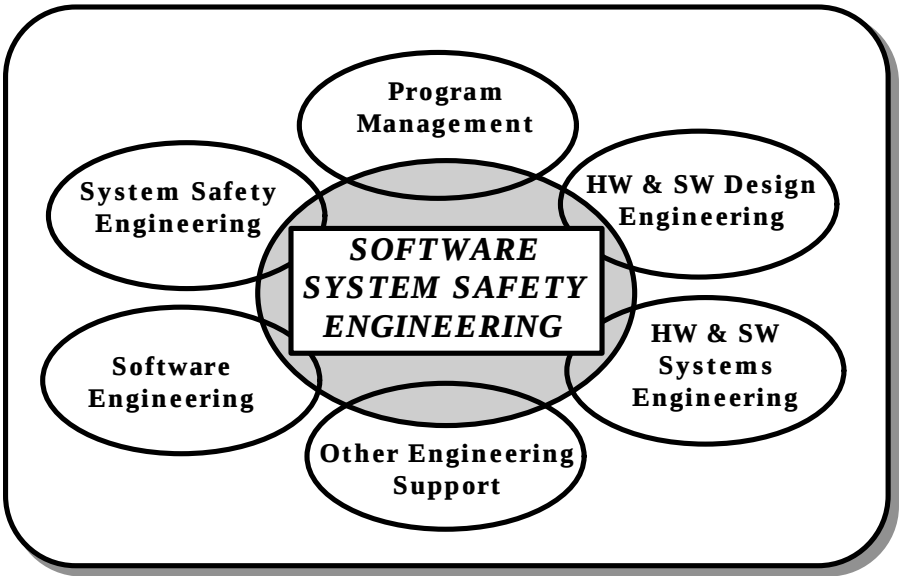


Figure 4-9: Software Safety Program Interfaces

4.2.1.3.1 MANAGEMENT INTERFACES

The program manager, under the authority of the Acquisition Executive (for DoD programs) or the Program Executive Officer (PEO):

- Coordinates the activities of each professional discipline for the entire program.
- Allocates program resources.
- Approves the programs' planning documents, including the System Safety Program Plan.

- Reviews safety analyses; accepts impact on system for Critical and higher category hazards (based upon acceptable levels of risk); and submits finding to Program Executive Office for acceptance of unmitigated, unacceptable, hazards.

It is the program managers responsibility to ensure that processes are in place within a program which meet, not only the programmatic, technical and safety objectives, but also the functional and system specifications and requirements of the customer. The Program Manager must allocate critical resources within the program to reduce social-political, managerial, financial, technical, and safety risk of the product being produced. Therefore, management support is essential to the success of the software system safety program.

The program manager ensures that the safety team develops a practical process and implements the necessary tasks required to identify system hazards and failure modes, perform causal factor analysis, derive design requirements to eliminate and/or control the hazards or failure modes, provide evidence for the implementation of design requirements, and analyze and assess the residual risk of any hazards that remain in the design at the time of system deployment and operation. The safety manager and the software engineering manager is dependent upon program management for the allocation of necessary resources (time, tools, training, money, and personnel) for the successful completion of the required software system safety engineering tasks.

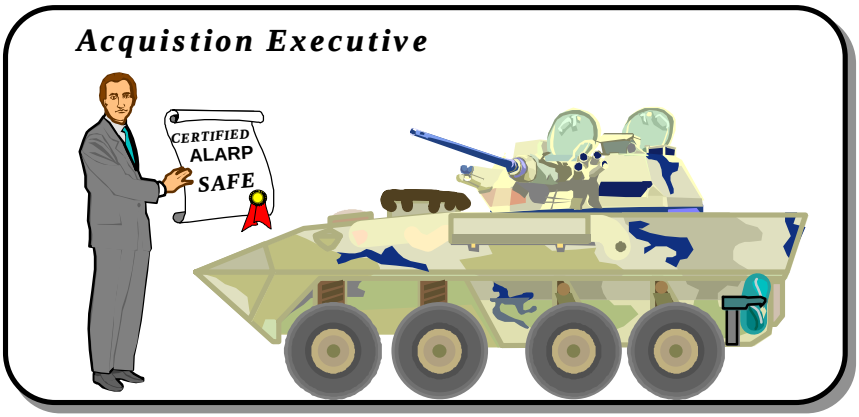


Figure 4-10: Ultimate Safety Responsibility

Within the DoD framework, the Acquisition Executive (Figure 4-10) is ultimately responsible for the acceptance of safety risk at the time of test, initial systems operation, and deployment. Using the development and test of a new weapon system as an example, the Acquisition Executive must certify at the Test Readiness Review (TRR), and the Safety Program Review Authority (SPRA) [sometimes referred to as a Safety Review Board (SRB)], that all hazards and failure modes have been eliminated or risk-reduced as-low-as-reasonably-possible (ALARP). An accurate assessment on the residual safety risk of a system, at this critical time, facilitates informed management and engineering decisions. Without the safety risk assessment provided by a credible system safety process, personal, professional, programmatic, and political liabilities would have to be assumed in the decision making process. The ability of the safety manager to provide an accurate assessment of safety risk is highly dependent upon the support provided by program management throughout the duration of the design and development of the system.

4.2.1.3.2 TECHNICAL INTERFACES

The engineering disciplines associated with system development must also provide technical support to the software safety engineering team (Figure 4-11). This essential engineering support must be provided by engineering management, design engineers, systems engineers, software development engineers, integrated logistics support and other support engineers. Other support engineers include, reliability, human factors, quality assurance, test & evaluation, verification & validation, and supportability. Each member of the engineering team must provide timely support to the defined processes of the software system safety team for the accomplishment of safety analyses and for specific design influence activities which eliminates, reduces, or controls hazard risk. This includes the traceability of software safety requirements from design-to-test with its associated and documented evidence of implementation.

A sure way to fail in the software safety activity is to fail to secure software engineering acceptance and support of the software safety process, functions and implementation tasks. One must recognize that most formal education and training for software engineering and software developers does not present, teach, or rationalize a safety engineering interface other than the incorporation of generic safety requirements. The system safety process relating to the derivation of functional safety requirements through hazards analyses is foreign to most software developers.

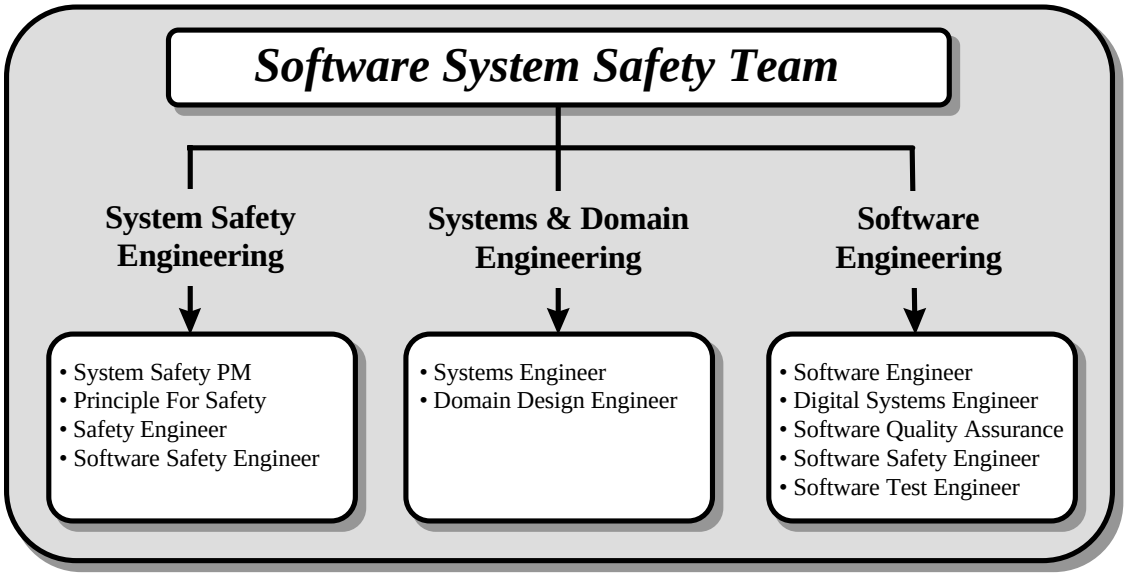


Figure 4-11: Proposed SSS Team Membership

Without the historical experience of cultivating technical interfaces between software development and system safety engineering, several issues may need to be resolved. They include:

- Software engineers may feel threatened that system safety has responsibility for activities considered to be under the responsibility and accountability of the software engineer.
- Software developers are confident enough in their own methods of error detection, error correction, and error removal, that they ignore the system safety inputs to the design process. This is normally in support of generic safety requirements.

- There is insufficient communication and resource allocation between software development and system safety engineering to identify, analyze, categorize, prioritize, and implement both generic and derived software safety requirements.

A successful software system safety effort can only be accomplished through the establishment of a technical software safety team approach. Team tasks must be defined and specific team expertise assigned responsibility and accountability for the accomplishment of these tasks. This expertise must be identified and defined in the software safety portion of the SSPP. The team must identify both the generic safety requirements and guidelines and the functional requirements derived from system hazards and failure modes that have specific software input or influence. Once these hazards and failure modes are identified, specific design and test requirements can be derived through an integrated effort. All software safety design requirements must be traceable to test and be correct, complete, and testable where possible. The implemented requirements must eliminate, control, or reduce the safety risk as low as reasonably possible while meeting the user requirements and specifications within operational constraints. Supplemental information pertaining to technical interfaces is found in Appendix C.

4.2.1.3.3 CONTRACTUAL INTERFACES

Management planning for the software system safety function includes the identification of contractual interfaces and obligations. It must be recognized that each program will have the potential to present unique challenges to the system safety and software development managers. This may include a Request For Proposal (RFP) which does not specifically address the safety of the system, to contract deliverables which are extremely costly to develop. Regardless of the challenges, the tasks needed to accomplish a software system safety program must be planned to meet both the system and user specifications and requirements and the safety goals of the program. The following are those contractual obligations that are deemed to be most essential for any given contract.

- Request For Proposal
- Statement of Work
- Contract
- Contract Deliverable Requirements List

4.2.1.4 CONTRACT DELIVERABLES

The Statement of Work defines the deliverable documents and products (Contract Deliverables Requirements List, (CDRL's)) desired by the customer. Each CDRL deliverable should be addressed in the System Safety Program Plan (SSPP) to include the necessary activities and process steps required to produce it. Completion of contract deliverables is normally tied to the acquisition life cycle of the system being produced and the program milestones identified in the System Engineering Master Plan (SEMP). The planning required by the system safety manager is to ensure that the system safety and software safety processes provide the necessary data and output for the successful accomplishment of the plans and analysis. The system safety schedule should track closely to the SEMP and be proactive and responsive to both the customer and the

design team. Contract deliverables should be addressed individually on the safety master schedule and within the SSPP.

As future procurements will likely not have specific military and DoD standards on contract, the procuring agency must ensure specific deliverables are identified and contractually required to meet programmatic and technical objectives. This activity must also specify the content and format of each deliverable item. As existing government standards become translated into commercial standards and guidance, the safety manager must ensure that sufficient planning is accomplished to specify the breadth, depth, and timeline of each deliverable (which is normally defined by Data Item Descriptions (DID's). The breadth and depth of the deliverable items must provide the necessary audit trail to ensure safety levels of risk are achieved (and are visible) during development, test, support transition, and maintenance in the out-years. The deliverables must also provide the necessary evidence or audit trail for validation and verification of safety design requirements. The primary method of maintaining a sufficient audit trail is the utilization of a developers safety data library. This library would be the repository for all safety documentation. Appendix C.1 describes the contractual deliverables which should be contained in the system safety data library.

4.2.1.5 DEVELOP SOFTWARE HAZARD CRITICALITY MATRIX

The ability to prioritize and categorize hazards is essential for the allocation of resources to the functional area possessing the highest risk potential. System safety programs have historically used the Hazard Risk Index (HRI) to categorize hazards. This concept was introduced in Section 3.6.1. However, the methodology to accurately categorize hazards using this traditional HRI matrix for hazards possessing software causal factors is insufficient. The ability to use the original (hardware oriented) HRI matrix was predicated on the probability of hazard occurrence and the ability to obtain component reliability information from engineering sources. The current technologies associated with the ability to accurately predict software error occurrence, and quantify its probability, is still in its development infancy. This is due to the nature of software as opposed to hardware. Statistical data may be used for hardware to predict failure probabilities. However, software does not fail in the same manner as hardware (it does not wear out, break, or have increasing tolerances). Software errors are generally requirements errors (failure to anticipate a set of conditions that lead to a hazard, or influence of an external component failure on the software) or implementation errors (coding errors, incorrect interpretation of design requirements). Therefore, assessing the risk associated with software is somewhat more complex. Without the ability to accurately predict a software error occurrence, supplemental methods of hazard categorization must be available when the hazard possesses software causal factors. This section of the handbook presents a method of categorizing hazards which possess software influence or causal factors.

4.2.1.5.1 HAZARD SEVERITY

Regardless of the hazard causal factors (hardware, software, human error, and software influenced human error) the severity of the hazard remains constant. This is to say that the consequence of hazard occurrence remains the same regardless of what actually caused the hazard to propagate within the context of the system. As the hazard severity is the same, the severity

table presented in Section 3.6.1.2 (Table 3-1, Hazard Severity), remains an applicable criteria for the determination of hazard criticality for those hazards possessing software causal factors.

4.2.1.5.2 HAZARD PROBABILITY

With the difficulty of assigning accurate probabilities to faults or errors within software modules of code, a supplemental method of determining hazard probability is required when software causal factors exist. Figure 4-12 demonstrates that in order to determine a hazard probability, software causal factors must be assessed in conjunction with the causal factors from hardware and human error. The determination of hardware and human error causal factor probabilities remain constant in terms of historical "best" practices. However, the likelihood of the software aspect of the hazard's cumulative causes must be addressed.

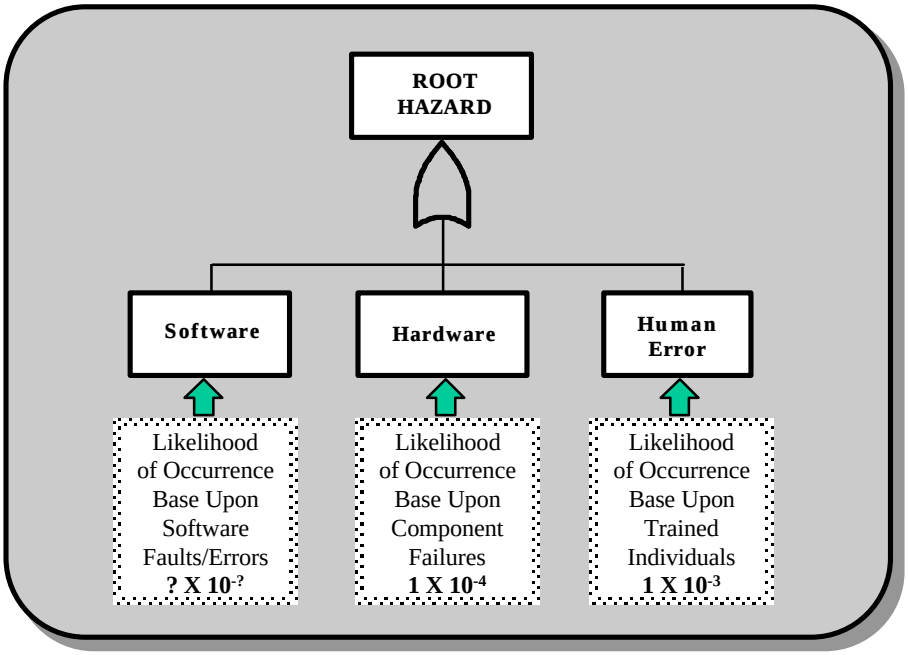


Figure 4-12: Likelihood of Occurrence Example

There have been numerous methods of determining the software's influence on system-level hazards. Two of the most popular are presented in MIL-STD 882C and RTCA DO-178B (see Figure 4-13). These do not specifically determine software-caused hazard probabilities, but instead assesses the software's "control capability" within the context of the

software causal factors. In doing so, each software causal factor can be labeled with a software control category for the purpose of helping to determine the degree of autonomy that the software has on the hazardous event. The software safety team must review these lists and tailor them to meet the objectives of the system safety and software development programs.

MIL-STD 882C	*RTCA-DO-178B*
(I) Software exercises autonomous control over potentially hazardous hardware systems, subsystems or components without the possibility of intervention to preclude the occurrence of a hazard. Failure of the software or a failure to prevent an event leads directly to a hazards occurrence.	**(A)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft.
(IIa) Software exercises control over potentially hazardous hardware systems, subsystems, or components allowing time for intervention by independent safety systems to mitigate the hazard. However, these systems by themselves are not considered adequate.	**(B)** Software whose anomalous behavior, as shown by the System Safety assessment process, would cause or contribute to a failure of system function resulting in a hazardous/severe-major failure condition of the aircraft.
(IIb) Software item displays information requiring immediate operator action to mitigate a hazard. Software failure will allow or fail to prevent the hazard's occurrence.	**(C)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a major failure condition for the the aircraft.
(IIIa) Software items issues commands over potentially hazardous hardware systems, subsystem, or components requiring human action to complete the control function. There are several, redundant, independent safety measures for each hazardous event.	**(D)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft.
(IIIb) Software generates information of a safety critical nature used to make safety critical decisions. There are several, redundant, independent safety measures for each hazardous event.	**(E)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of function with no effect on aircraft operational capability or pilot workload. Once software has been confirmed as level E by the certification authority, no further guidelines of this document apply.
(IV) Software does not control safety critical hardware systems, subsystems, or components and does not provide safety critical information.	

Figure 4-13: Examples of Software Control Capabilities

Once again, the concept of labeling software causal factors with control capabilities is foreign to most software developers and programmers. They must be convinced that this activity has utility in the identification and prioritization of software entities which possesses safety implication. In most instances, the software development community desires the list to be as simplistic and short as possible. The most important aspect of the activity must not be lost, that is, the ability to categorize software causal factors for the determining of both hazard likelihood, and the design, code, and test activities required to mitigate the potential software cause. Autonomous software with functional links to catastrophic hazards demand more coverage than software that influences low-severity hazards.

4.2.1.5.3 SOFTWARE HAZARD CRITICALITY MATRIX

The Software Hazard Criticality Matrix (SHCM) assists the software safety engineering team and the subsystem and system designers in allocating the software safety requirements between software modules and resources, and across temporal boundaries (or into separate architectures). The software control measure of the SHCM also assists in the prioritization of software design and programming tasks.

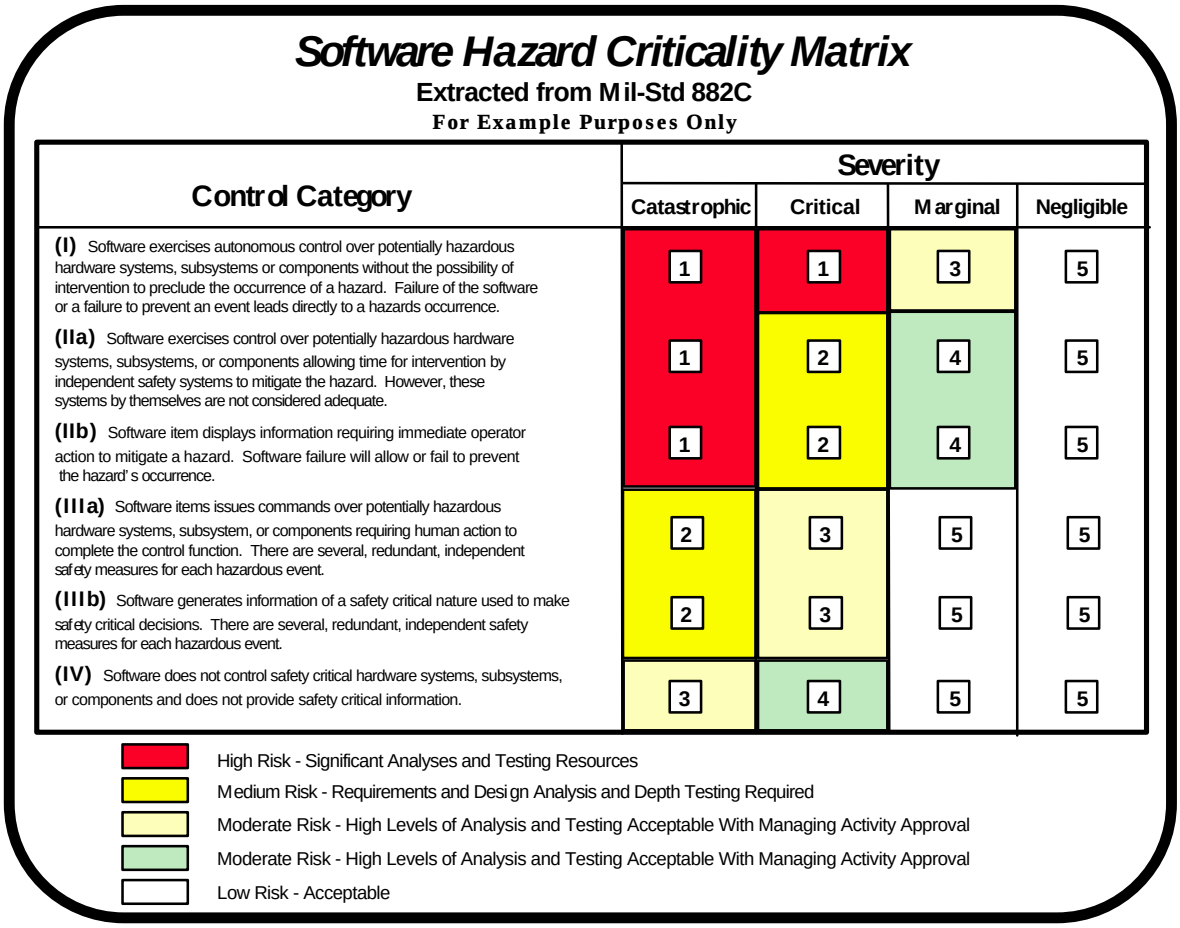


Figure 4-14: Software Hazard Criticality Matrix -882C

4.2.2 MANAGING

Software systems safety program management (Figure 4-15), like system safety program management, begins early in the system concept, as soon as the system safety program is established, and continues throughout the system development. Management of the effort requires a variety of tasks or processes, from establishing the Software Safety Working Group to preparing the System Safety Assessment Report. Even after a system is placed in service, management of the software systems safety effort continues to address modifications and enhancements to the software and the system. Often, changes in the use or application of a system necessitate a re-assessment of the safety of the software in the new application.

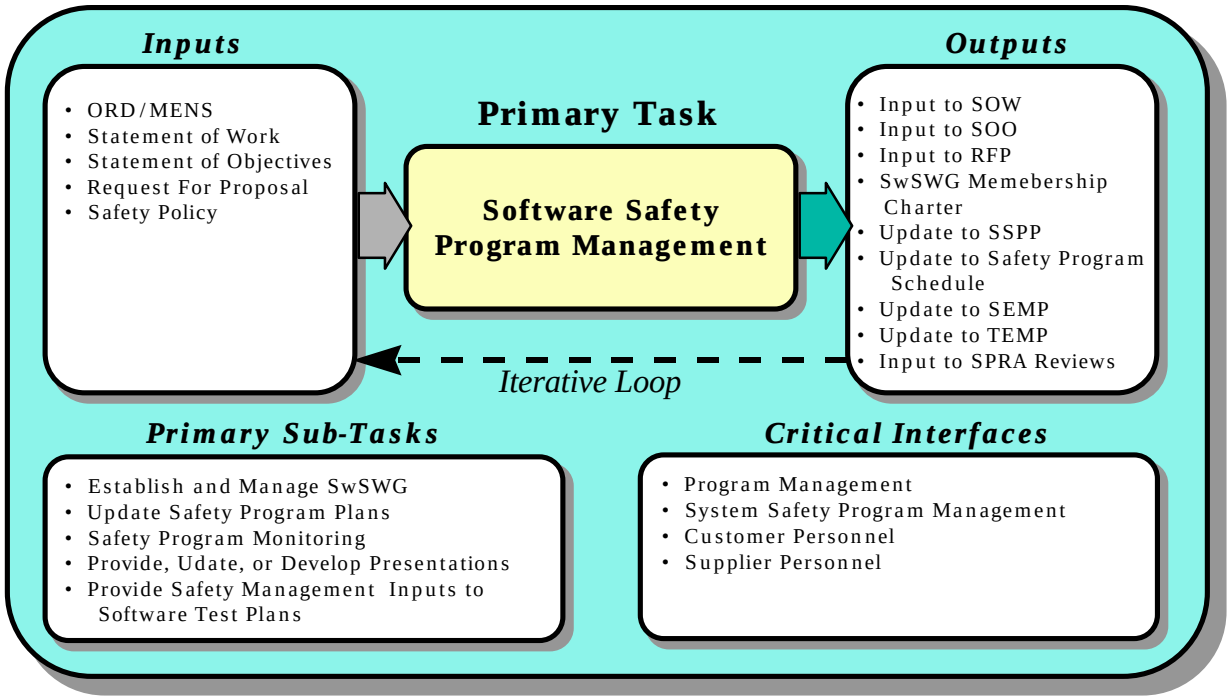


Figure 4-15: Software Safety Program Management

Effective management of the safety program is essential to the effective and efficient reduction of system risk. This section discusses the management aspects of the software safety tasks and provides guidance in establishing and managing an effective software safety program. Initiation of the system safety program is all that is required to begin the activities pertaining to software safety tasks. Initial management efforts parallel portions of the planning process since many of the required efforts (such as establishing a hazard tracking system or researching lessons learned) need to begin very early in the safety program. Safety management pertaining to software generally ends with the completion of the program and its associated testing, whether it is a single phase of the development process (e.g., concept exploration), or continues through the development, production, deployment, and maintenance phases. In the context of acquisition reform, this means that management of the efforts must continue throughout the system life cycle. From a practical standpoint, management efforts end when the last safety deliverable is completed and is accepted by the customer. Management efforts then may revert to a "caretaker" status in which the safety manager monitors the use of the system in the field and identifies potential safety deficiencies based on user reports and accident/incident reports. Even if the developer has no responsibility for the system after deployment, the safety program manager can develop a valuable database of lessons learned for future systems by identifying these safety deficiencies.

The establishment of a software safety program includes the establishment of a Software Safety Working Group (SwSWG). This is normally a sub-group of the SSWG and chaired by the PFS. The SwSWG has overall responsibility for:

- Monitoring and control of the software safety program
- Identifying and resolving hazards with software causal factors
- Interfacing with the other IPT's

- And, performing final safety assessment of the system design.

A detailed discussion of a SwSWG is found in the supplemental information of Appendix C.5.2.

It is in this phase of the program that the Plan of Actions and Milestones (POA&M) is developed and based on the overall software development program POA&M and in coordination with the System Safety POA&M. Milestones from the software development POA&M, particularly design reviews and transition points (e.g., from unit code and test to integration) determine the milestones required of the software safety program. The SwSWG must ensure that the necessary analyses are complete in time to provide the necessary input to various development efforts to ensure effective integration of software safety into the overall software development process. The overall Phases, Milestones and Processes Chart (Figure 4-22) identifies the major program milestones from MIL-STD-498 and 499 with the associated software safety program events.

One of the most difficult aspects of the software safety program management is the identification and allocation of resources required to adequately assess the safety of the software. In the early planning phases, the configuration of the system and the degree of interaction of the software with the potential hazards in the system is largely unknown. The higher the degree of software involvement, the greater the resources required to perform the assessment. To a large extent, the software safety program manager can use the early analyses of the design, participation in the functional allocation, and high level software design process to ensure that the amount of safety-critical software is minimized. If safety-critical functions are distributed throughout the system and its related software, then the software safety program must encompass a much larger portion of the software. However, if the safety-critical functions are associated with as few software modules as practical, the level of effort may be significantly reduced.

Effective planning and integration of the software safety efforts into the other integrated process teams will significantly reduce the software safety related tasks that must be performed by the SSS Team. Incorporating the generic software safety requirements into the plans developed by the other IPT's allows them to assume responsibility for their assessment, performance and/or evaluation. For example, if the SSS Team provides the quality assurance generic requirements to the Software Quality Assurance IPT, they will perform compliance assessments with requirements, not just for safety, but for all aspects of the software engineering process. In addition, if the SQA IPT “buys-into” the software safety program and its processes, it can significantly supplement the efforts of the software safety engineering team. The same is true of the other IPT’s such as configuration management and software test and evaluation. In identifying and allocating resources to the software safety program, the software safety program manager can perform advance planning, establish necessary interfaces with the other IPT’s, and identify individuals to act as software safety representatives on those IPT’s.

Identifying the number of analyses and the level of detail required to adequately assess the software involves a number of processes. Experience with prior programs of a similar nature is the most valuable resource that the software safety program manager has for this task. However, every program development is different and involves different teams of people, procuring agency requirements, and design implementations. The process begins with the identification of the system level hazards in the Preliminary Hazards List. This provides a gross idea of the concerns that must be assessed in the overall safety program. From the system specification review process, the functional allocation of requirements results in a high level distribution of safety-

critical functions and system level safety requirements to the hardware, software, operator, and maintenance. The safety-critical functions and requirements are thus known in general terms. Software functions that have a high safety-criticality (e.g., warhead arming and firing) will require a significant analysis effort that may include code level analysis. Safety's early involvement in the design process can reduce the amount of software that requires analysis, however, the software safety manager must still identify and allocate resources to perform these tasks. Those in which the safety requirements may conflict with other (e.g., reliability) requirements require trade-off studies be performed to achieve a balance between all desirable attributes. The safety manager must include the conduct of these trade studies in the planning process and subsequently identify and allocate all necessary resources to accomplish the task.

The software control categories discussed in Section 4.2.1.5 provide a useful tool for identifying software that requires high levels of analysis and testing. Obviously, the higher the HRI, the higher the level of effort required to analyze, test, and assess the risk associated with the software. In the planning activities, the SwSWG identifies the analyses necessary to assess the safety of specific modules of code. Experience is the best teacher in determining the level of effort required to perform this analyses. The analyses do not need to be performed by the software engineering group themselves, it may be assigned to another group or person with the specialized expertise required to perform the analysis. The SwSWG will have to provide the necessary safety related guidance and training to the individuals performing the analysis, but only to the extent necessary for them to accomplish the task.

One of the most important aspects of software safety program management is monitoring the activities of the safety program throughout system development to ensure that tasks are on schedule and within cost, and to identify potential problem areas that could impact the safety or software development activities. The software safety manager must:

- Monitor the status and progress of the software and system development effort to ensure that program schedule changes are reflected in the software safety program POA&M.
- Monitor the progress of the various IPT's and ensure that the safety interface to each is working effectively. When problems are detected, either through feedback from the software safety representative or other sources, the software safety manager must take the necessary action to mitigate the problem.
- Monitor and receive updates regarding the status of analyses, open HAR's, and other safety activities on a weekly basis. Significant HAR's should be discussed at each SwSWG meeting and the status updated as required. A key factor that the software safety program manager must keep in mind, is the tendency for many software development efforts to begin compressing the test schedule as slippages occur in the software development schedule. He or she must ensure that the safety test program is not compromised as a result of these slippages.

The SPRA requirements vary with the procuring agency and are often governed by procuring agency directives. Generally, the contract will identify the review requirements, however, it is the responsibility of the developing agency to ensure that the software safety program incorporate the appropriate reviews into the software safety program plans. The system safety manager must identify the appropriate SPRA and review the schedule during the development process. SPRA

reviews generally involve significant effort outside the other software safety tasks. The developing agency must determine the level of effort required for each review, the support that will be required during the review, and incorporate these into the Software Safety Program Plan (SwSPP). For complex systems, multiple reviews are generally required to update the SPRA and ensure that all of the procuring agency's requirements are achieved.

Although SPRA requirements may vary from each procuring agency, some require a technical data package and briefing to a review board. The technical data package may be a safety assessment report or may be considerably more complex. The developing agency must determine whether they are to provide the technical data package and briefing, or whether that activity is to be performed independently. In either event, safety program personnel may be required to participate or attend the reviews to answer specific technical questions that may arise. Normally, the presenters will require several weeks of preparation for the SPRA reviews. Preparation of the technical data package and supporting documentation requires time and resources even though the data package is a draft or final version of the Safety Assessment Report (SAR).

4.3 SOFTWARE SAFETY TASK IMPLEMENTATION

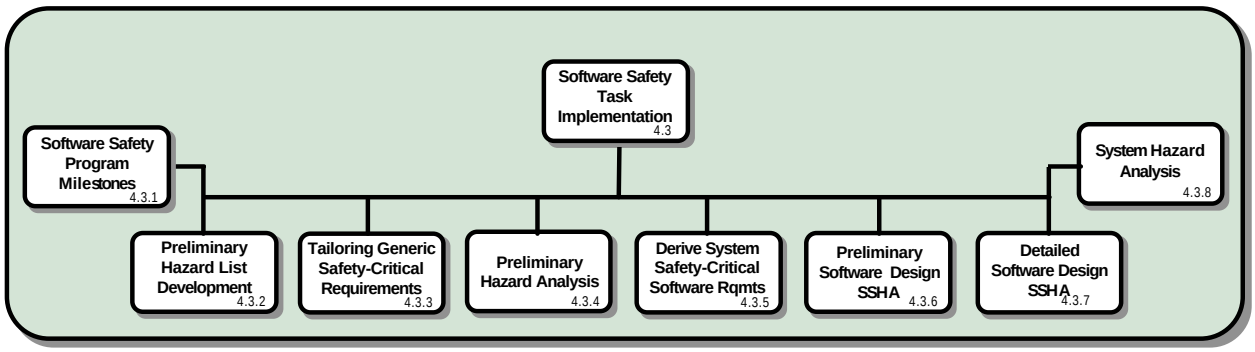


Figure 4-16: Software Safety Task Implementation

This section of the handbook describes the primary task implementation steps required for a baseline software safety engineering program. It presents the necessary tasks required for the integration of software safety activities into the functional areas of system and software development. Remember, software system safety (or software safety) is a defined **subset** of both the system safety engineering process and the software engineering and development process.

As the software safety engineering process is being introduced, inputs to the described tasks and the products that the specific process step produces will be identified. Each program and engineering interface tied to software safety engineering must be in agreement with the processes, tasks, and products of the software safety program and must agree with the timing and scope of effort to verify that it is in concert with the objectives and requirements of each interface. If other program disciplines are not in agreement, or do not see the functional utility of the effort, they will usually default to a "non-support" mode.

Figure 4-16 provides a graphical depiction of the software safety activities required for the implementation of a credible software safety program. Remember, the process steps identified in this handbook represent a baseline program that has a historical lessons learned base and includes

best practices from successful programs. As each procurement, software acquisition, or development has the potential and probability to be uniquely diverse, the safety manager *must* use this section as a guide only. Each of the following steps should be analyzed and assessed to identify where minor changes are required or warranted for the software development program proposed. If these tasks, with the implementation of minor changes, are incorporated in the system acquisition life cycle, the software system safety effort has a very high likelihood of success.

In addition, the credibility of software safety engineering activities within the hardware and software development project is predicated on the credibility of the individual(s) performing the managerial and technical safety tasks, and the identification of a logical, practical, and economical process which produces the safety products to meet the safety objectives of the program. In this context, the primary safety products include hazards analysis, safety requirements for implementation into the design, test requirements to produce evidence for the elimination and/or control of the safety hazards, and the identification of safety requirements pertaining to operations and support of the product. The software safety tasks as defined in this section, must be agreed to by managerial and technical interfaces, and provide the documented evidence for the resolution of identified hazards and failure modes in design, manufacture (code in software), fabrication, test, deployment, and support activities. It must also thoroughly define and communicate residual safety risk to program management, at any point in time, during each phase of the development life cycle.

4.3.1 SOFTWARE SAFETY PROGRAM MILESTONES

The planning and management of a successful software safety program is supplemented by the safety engineering and management program schedule. The schedule should include near-term and long-term events, milestones, and contractual deliverables. The schedule should also reflect the system safety management and engineering tasks that are required for each life cycle phase of the program and to support DoD milestone decisions. Also of importance, is specific safety data that is required to support special safety boards that may be required for compliance and certification purposes. Examples include, but are not limited to, FAA certification, DoD Weapon Systems Explosive Safety Review Board (WSESRB), DNS Nuclear Certification, and the Non-Nuclear Munitions Safety Board. Each event, deliverable, and/or milestone should be tracked to ensure suspense's and safety analysis activities are timely in the development process to help facilitate cost-effective and technically feasible design solutions. These activities ensure that the software safety program will meet the desired safety specifications of program and the system development activities.

Planning for the system safety program must include the allocation of resources to support travel of safety management and engineers. The contractual obligations of the Statement of Work in concert with the processes stated in the program plans, and the required support of program meetings, dictate the scope of safety involvement. With the limited funds and resources of today's programs, the system safety manager must determine and prioritize the level of support that will be allocated to program meetings and reviews. The number of meetings that require support, the number of safety personnel that are scheduled to attend, and the physical location of the meetings must be assessed against the budgeted travel allocations for the safety function. The

resource allocation activity becomes complicated if priorities are not established “up-front” with the determination of meetings to support.

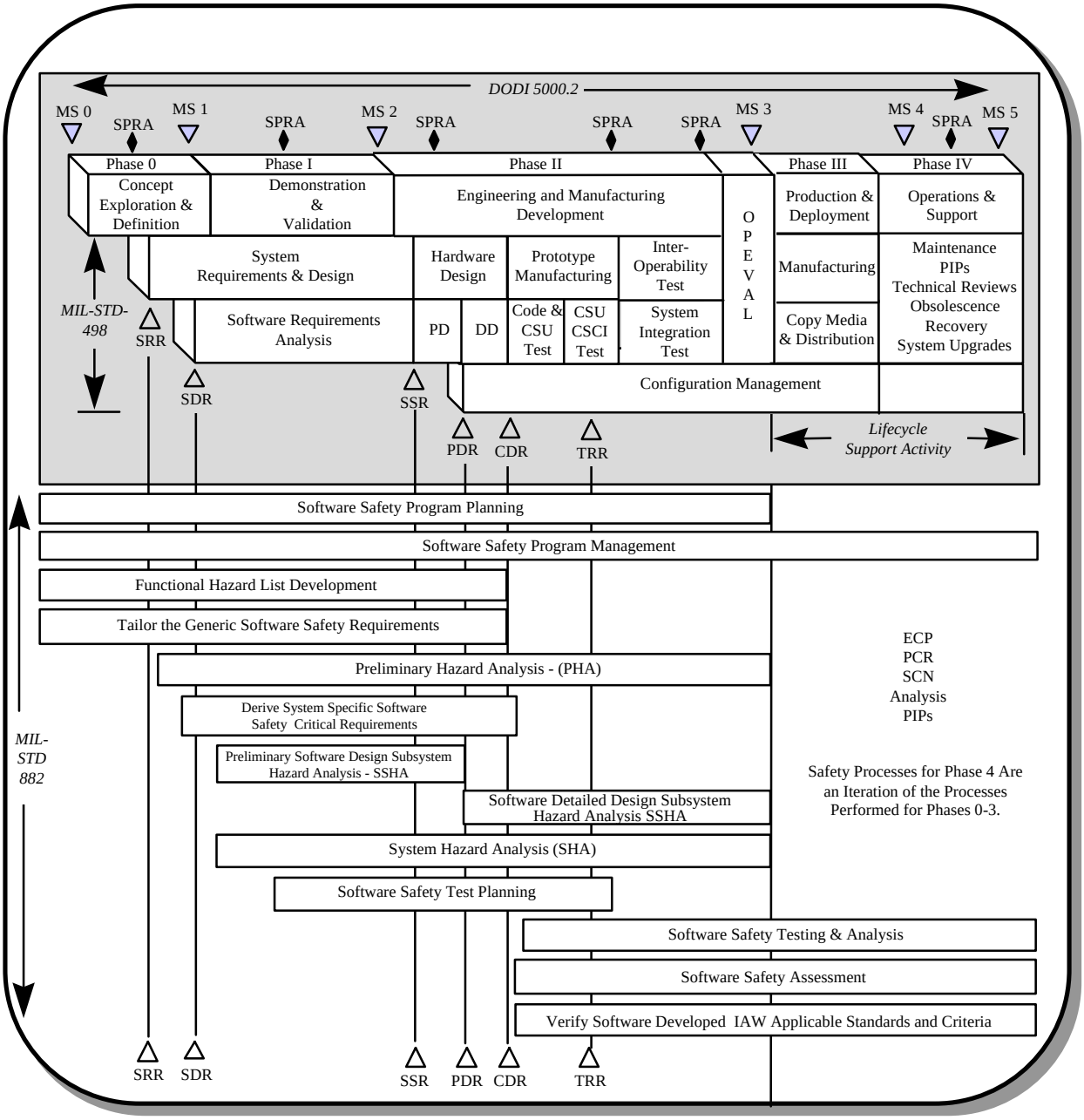


Figure 4-17: Software Safety Program Schedule

Once priorities are established, meetings that cannot be supported due to budget constraints, can be communicated to program management for the purpose of concurrence, or the reallocation of resources with program management direction. Figure 4-17 provides an example milestone schedule for a software safety program. It graphically depicts the relationship of safety specific activities to the acquisition life cycles of both system and software development.

Remember, each procurement is unique and may have subtle differences associated with managerial and technical interfaces and timelines. This schedule must be used as an example where specific activities and time relationships are based on "generic" or "typical" programs. Program specific differences must be integrated into the schedule and must support the practical assumptions and limitations of the program.

As described in Section 4.2.2, the POA&M will also include the various safety reviews, procuring agency reviews, internal reviews, and the SwSWG meetings. The software safety assessment milestones are generally geared to the SPRA reviews, since the technical data package required, is in fact, either the draft or final software safety assessment report. Hazard analysis schedules must reflect program milestones where hazard analysis input is required. For example, safety design requirements from generic requirements tailoring (documented in the Safety Requirements Criteria Analysis (SRCA)) must be available as early as practical in the design process for integration into design documentation, programmatic documents, and the system safety documents. Specific safety requirements from the Preliminary Hazards Analysis (PHA) must be available prior to the Preliminary Design Review (PDR) for integration into the design documents. System safety and software safety must participate in the system specification review and provide recommendations during the functional allocation of system requirements to hardware, software, operations and maintenance. After functional allocation is complete, the software engineering IPT, with the help of the software safety representative, will develop the Software Requirements Specifications (SRS). The SwSWG updates the analyses as the system development progresses, however, the requirements must be complete prior to the Critical Design Review (CDR). Requirements added after the CDR can have a major impact on program schedule and cost.

The preliminary design analyses assess the system and software architecture, and provide design recommendations to reduce the associated risk. During the development of the SRS, the SSS Team initiates the Subsystem Hazards Analysis (SSHA) and its assessment of preliminary software design for this purpose. This analysis provides the basis for input to the design of the Computer Software Configuration Items (CSCI's), and the individual software modules. As the design progresses and detailed specifications are available, the SSS Team initiates SSHA which assesses the detailed software design. The team analyzes the design of each module containing safety-critical functions. For highly critical software, the analysis will extend to the source code to ensure that the intent of the safety requirements is properly implemented.

Development of safety test requirements begins with the identification of safety design requirements. This information must be provided to the testing organization in time for preparation of test plans and test scenarios. Detailed inputs regarding specific safety tests are derived from the hazard analyses, causal factor analysis, and the definition of software hazard mitigation requirements. Safety-specific test requirements are provided to the test organization for development of specific test procedures to validate the safety requirements. The analysis associated with this phase begins as soon as test data from the safety tests is available.

The Systems Hazards Analysis (SHA) begins as soon as functional allocation of requirements occurs and progresses through the completion of system design. Specific milestones for the SHA include providing safety test requirements for integration testing to the test organization and detailed test requirements for interface testing. The latter will be required before testing of the software with other system components begins.

4.3.2 PRELIMINARY HAZARD LIST (PHL) DEVELOPMENT

The PHL is a contractual deliverable on typical programs and is described in Appendix C.1.3. This is the initial cutset of hazards associated with the system under development. Development of the PHL requires a knowledge of the physical and functional requirements of the system and some foreknowledge of the conceptual system design. The documentation of the PHL helps to initiate the analysis that must be performed on the system, subsystems and their interfaces. This list is based upon the review of analysis of similar systems, lessons learned, potential kinetic energies associated with the design, design handbooks, and user and systems specifications. The list also aids in the development of generic (or preliminary) requirements for the system designers and the identification of programmatic (technical or managerial) risks to the program.

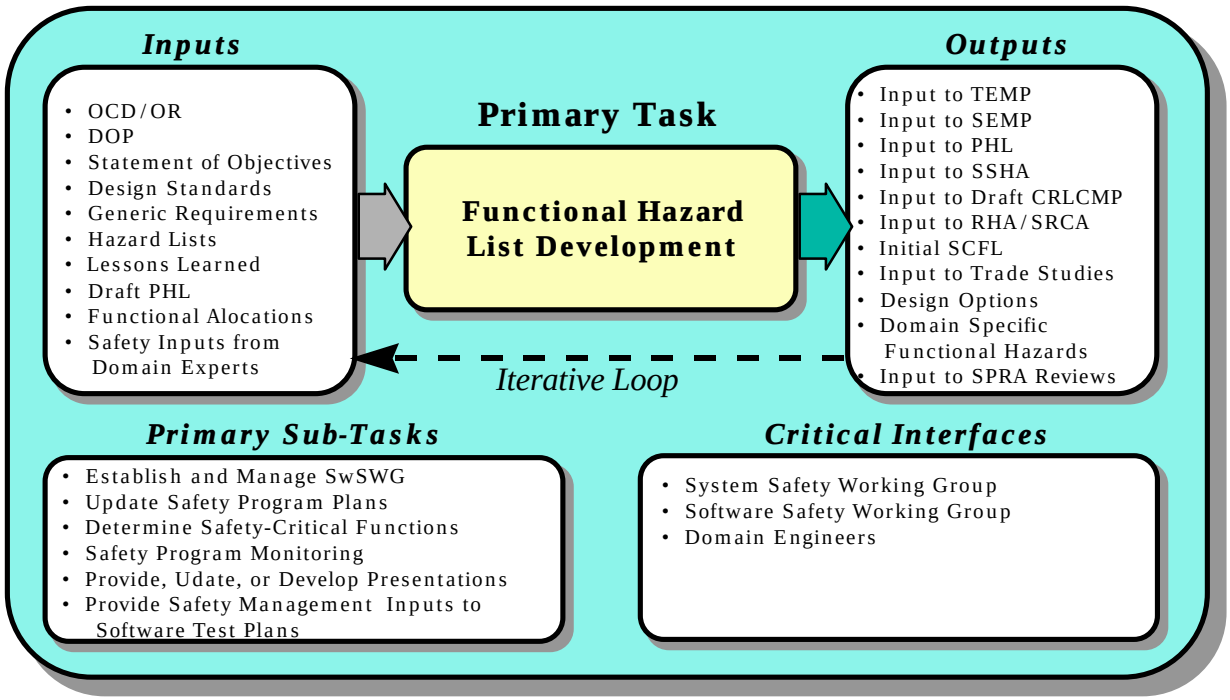


Figure 4-18: PHL Development

The development of the PHL is an integrated engineering task which requires cooperation and communication between functional disciplines and among systems, safety, and design engineers. This task is accomplished by the assessment and analysis of all preliminary and current data pertaining to the proposed system that can be gathered. From a documentation perspective, the following should be available for review:

- Preliminary system specification
- Preliminary product specification
- User requirements document
- Lessons Learned
- Analysis of similar systems
- Design criteria and standards

From the preceding list of documentation and functional specifications, a preliminary list of system hazards is developed for further analysis. Although the identified hazards may appear at this time to be generic or immature, this is normal for the early phase of system development. As the hazards are analyzed against system physical and functional requirements they will mature to become the hazards fully documented in the PHA, SSHA, SHA and O&SHA. The preliminary assessment of the PHL hazards will help determine whether trade studies or design options must be considered to reduce the potential for unacceptable or unnecessary safety risk in the design.

SAFETY-CRITICAL FUNCTIONS
**** for example purposes only ****

- Altitude Indication
- Attitude Indication
- Air Speed Indication
- Engine Control
- Inflight Restart After Flameout
- Engine Monitor and Display
- Bleed Air Leak Detection
- Engine/APU Fire Detection
- Fuel Feed for Main Engines
- Fire Protection/Explosion Suppression
- Flight Control - Level III Flying Qualities
- Flight Control - Air Data
- Flight Control - Pitot Heat
- Flight Control - Fuel System/CG Control
- Flight Control - Cooling
- Flight Control - Electrical
- Flight Control - Hydraulic Power
- Canopy Defog
- Adequate Oxygen Supply to the Pilot
- Stores and/or Expendables Separation
- Safe Gun and Missile Operation
- Armament/Expendables for System Ground Operations
- Emergency Canopy Removal
- Emergency Egress
- Ejection Capability
- Landing Gear Extension
- Ground Deceleration
- Structure Capability to Withstand Flight Loads
- Freedom From Flutter
- Stability in Pitch, Roll and Yaw
- Heading Indication
- Fuel Quantity Indication
- Fuel Shut-off to Engine and APU
- Engine Anti-Ice
- Caution and Warning Indications

Figure 4-19: Safety-critical Functions - An Example

In addition to the information assessed from preliminary documents and databases, technical discussions should be held to determine the ultimate functions of the system that would be deemed safety-critical. Functions that should be assessed include manufacture, fabrication, operations, maintenance and test. Other technical considerations that must be assessed and analyzed include transportation & handling, software/hardware interfaces, software/human interfaces, hardware/human interfaces, environmental health & safety, explosive constraints, product loss prevention, and nuclear safety considerations.

This effort begins with the safety engineer analyzing the functionality of each segment of the conceptual design. From the gross list of system functions, the analyst must determine the safety ramifications of loss of function, interrupted function, incomplete function, or the function occurring out of sequence. This activity also provides for the initial identification of safety-critical functions. The rationale for the identification of safety-critical functions of the system is addressed in the identification of safety deliverables (Appendix C.1.4). It should be reiterated at this point, that this is an activity that must be performed as a part of the defined software safety

process. This process step ensures that the project manager, systems & design engineers, and the software developers & engineers are aware of each function of the design which is considered safety-critical or to have a predefined level of safety impact. It also ensures that each individual module of code that performs these functions is officially labeled as “safety-critical” and that defined levels of design, code, and test activity is mandated. An example of the possible safety-critical functions of a tactical aircraft is provided in Figure 4-19.

The benefit of identifying the safety-critical functions of a system is two-fold. First, it assists the software safety team in the categorization and prioritization of safety requirements for the software architecture early in the design life cycle. If the software performs or influences the safety-critical function(s), that module of code becomes safety-critical by default. This eliminates emotional discussions on whether individual modules of code are designed and tested to specific and extensive criteria. Second, it reduces the level of activity and resource allocations to software code not identified as safety-critical. This benefit is cost avoidance.

At this phase of the program, specific ties from the preliminary hazard list to the software design is quite premature. Specific ties to the software are normally through hazard causal factors and these have yet to be defined at this point in the development. However there may be identified hazards which have preliminary ties to safety-critical functions which in turn are functionally linked to the preliminary software design architecture. If this is the case, this functional link should be adequately documented in the safety analysis for further development and analysis. At the same time there are likely to be specific “generic” software safety requirements which are applicable to the system. These requirements are available from multiple sources and must be specifically tailored for the program as they apply to the system design architecture.

4.3.3 TAILORING GENERIC SAFETY-CRITICAL REQUIREMENTS

Figure 4-20 depicts the software engineering process for tailoring the generic safety-related software requirements list. Generic software safety requirements are those design features, development processes, "best practices", coding standards and techniques, and other general requirements that are levied on a system containing safety-critical software, regardless of the functionality of the application. The PHL, as described above, may help determine the disposition or applicability of many individual generic requirements. The software safety analysis must identify the applicable generic software safety requirements necessary to support the development of the Software Requirement Specification (SRS). A tailored list of these requirements should be provided to the developer for inclusion into the software requirements document.

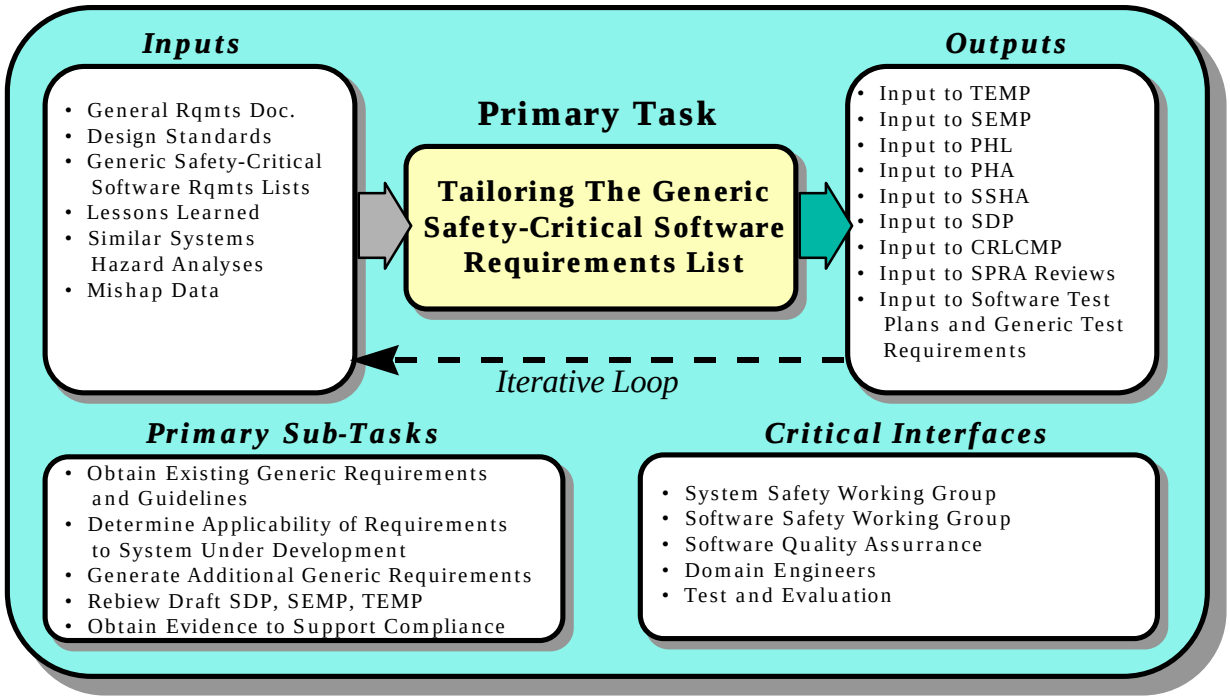


Figure 4-20: Tailoring the Generic Safety Requirements

Lists of generic safety requirements for consideration have been published by several individuals, agencies, and/or institutions. To date the most thorough is included in Appendix D, Generic Requirements and Guidelines, which includes the STANAG 4404, NATO Standardization Agreement, *Safety Design Requirements and Guidelines for Munitions Related Safety-Critical Computing Systems*, the Mitre (Ada) list, and other language specific requirements. These requirements should be assessed and prioritized according to applicability to the development effort. Whatever list is used, the analyst must assess each item individually for compliance, non-compliance, or non-applicability. On a particular program, the agreed upon generic software safety requirements should be included in the Safety Requirements Criteria Analysis (SRCA) and appropriate high level system specifications.

Figure 4-21 is an example worksheet form that may be used to track generic safety-critical software requirements implementation. If the program is complying with the requirement, the physical location of the requirement and the physical location of the evidence of implementation must be cited in the EVIDENCE block. If the program is not complying with the requirement (i.e., too late in the development to impose a safety kernel) or the requirement is not-applicable (i.e., an Ada requirement when developing in Assembler), a statement of explanation must be included in the RATIONALE block. An alternative mitigation of the source risk that the requirement addresses should be described if applicable, possibly pointing to another generic requirement on the list.

GENERIC SAFETY-CRITICAL SOFTWARE REQUIREMENTS IMPLEMENTATION	INTENDED COMPLIANCE		
	YES	NO	N/A
Item: Coding Requirements Issues Has an analysis (scaling, frequency response, time response, discontinuity, initialization, etc.) of the macros been performed?		X	
Rationale: (If NO or N/A, describe the rationale for the decision and resulting risk.) There are no macros in the design (discussed at checklist review 1/11/96)			
Evidence: (If YES, describe the kind of evidence that will be provided. Note: Specify sampling percentage per SwSPP, if applicable.)			
Action: (State the Functional area with responsibility.) Software Development POC:			

Figure 4-21: Example Generic Software Safety Requirements Tracking Worksheet

Caution should be given regarding the "blanket" approach of establishing the entire list of guidelines/requirements for a program. It must be remembered that each requirement will cost the program an expenditure of critical resources; people to assess and implement; budget for the design, code, and testing activities; and program schedule. Thus, these requirements should be assessed and prioritized according to applicability to the development effort. Inappropriate requirements which have not been adequately assessed are unacceptable. The analyst must assess each requirement individually and introduce only those which may apply to the development program.

Some requirements only necessitate a sampling of evidence to provide implementation (i.e., no conditional go-to's). These decisions should be made by the safety team. The lead software developer will often be the appropriate individual to gather the implementation evidence of the generic software-safety-critical requirements from the team member who holds or can provide the evidence. He may assign quality assurance, configuration management, validation & verification, human factor, software designers, or systems designers to fill out individual worksheets.

The entire tailored list of completed forms should be approved by the system safety engineer and thus submitted to the Safety Data Library (SDL) referred to by the Safety Assessment Report as evidence of generic software-safety-critical requirement implementation.

4.3.4 PRELIMINARY HAZARD ANALYSIS

The PHA is a safety engineering and software safety engineering analysis that is performed to identify the priority hazards and their casual factors in the system under development. Figure 4-22

depicts the safety engineering process for the PHA (there is nothing unique about the software aspects other than where the causal factors are located).

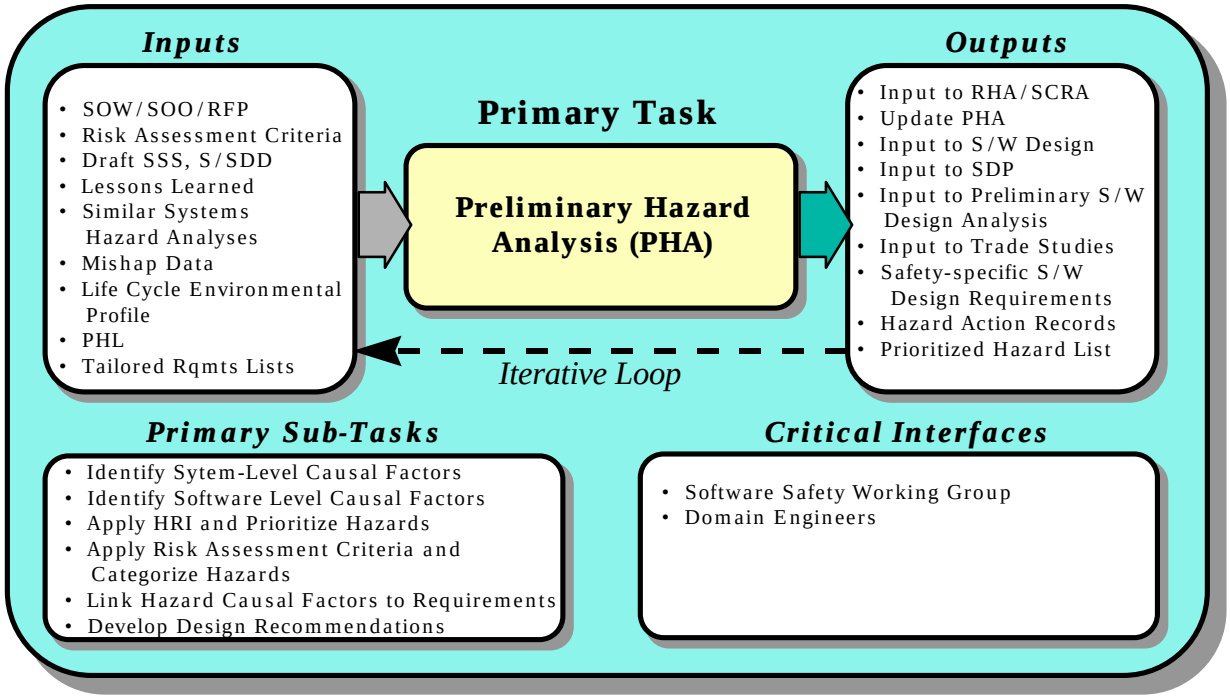


Figure 4-22: Preliminary Hazard Analysis

The PHA becomes the spring-board documentation to launch the subsystem and system hazard analysis as the design matures and it progresses through the development life cycle. Preliminary hazards can be eliminated (or officially closed through the SSWG) if they are deemed to be inappropriate for the design. Remember, this analysis is preliminary and is used to provide early design considerations which may, or may not, be derived or matured into design requirements. It is acceptable to eliminate hazards that are not applicable to the design

Throughout this analysis the PHA is used to provide input to trade-off studies. Trade-off analyses performed in the acquisition process are listed in Table 4-1 [DSMC, 1990]. These analyses are used on development programs to offer alternative considerations for performance, producability, testability, survivability, compatibility, supportability, reliability, and system safety during each phase of the development life cycle. System safety inputs to trade studies include the identification of potential safety concerns, and recommendations of credible alternatives which may meet all (or most) of the requirements while reducing overall safety risk.

The entire unabridged list of potential hazards developed in the PHL should be the entry point of PHA. The list should be “scrubbed” for applicability and reasonableness as the system design progresses. The first step is to eliminate from the PHL any hazards not applicable to the system. The remaining list of hazards should be categorized according to the (System) Hazard Risk Index and then prioritized according to this categorization. This provides an initial assessment of system hazard severity and probability of occurrence. The probability assessment at this point is usually subjective and qualitative.

Acquisition Phase	Trade-Off Analysis Function
Mission Area Analysis	Prioritize Identified User Needs
Concept Exploration	Compare New Technologies With Proven Concepts Select Concepts Best Meeting Mission Needs Select Alternative System Configurations
Demonstration Validation	Select Technology Reduce Alternative Configurations to a Testable Number
Full Scale Development	Select Component/Part Designs Select Test Methods Select Operational Test & Evaluation Quantities
Production	Examine Effectiveness of all Proposed Design Changes Perform Make-Or-Buy, Process, Rate, and Location Decisions

Table 4-1: Acquisition Process Trade-Off Analyses

After the prioritized list of preliminary hazards to be analyzed is determined, the analysis proceeds with determining the hardware, software, and human interface causal factors to the individual hazard as shown in Figure 4-23.

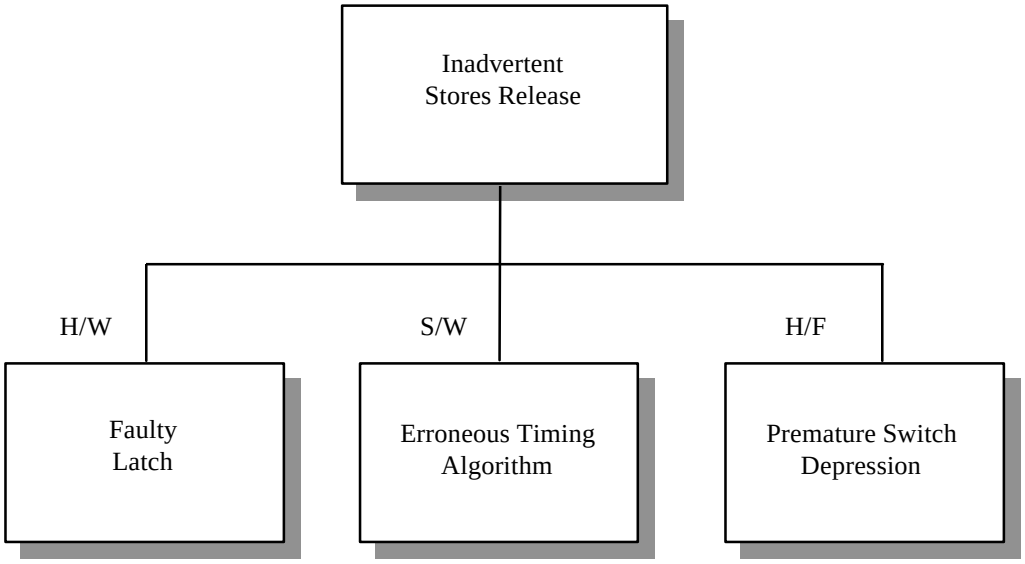


Figure 4-23: Hazard Analysis Segment

This differentiation of causes assists in the separation and derivation of specific design requirements that are attributed to software. As the analysis progresses, for example, a hardware casual factor could subsequently be contributed to by software or hardware. A hardware component failure may cause the software to react in an undesired manner leading to a hardware-influenced software causal factor. All paths must be considered to ensure coverage of the software safety analysis.

Although this tree can represent the entire system, software safety is particularly concerned with the software causal factors link to individual hazards and ensuring the mitigation of each causal factor is traced from requirements to design, code, and is subsequently tested. These preliminary analyses and subsequent system and software safety analyses identify when software is a potential cause of a hazard, or will be used to support the control of a hazard.

At this point, trades evolve. It should become apparent at this time whether hardware, software, or human training best mitigates the first level causal factors of the PHL item (the root event that is undesirable). This causal factor analysis begins to initiate insight into the best functional allocation and software design. It should be noted at this time, that requirements which are defined to mitigate hazard causal factors do not have to be one-to-one. That is, one software causal factor generating one software control requirement. Safety requirements can be one-to-one, one-to-many, or many-to-one in terms of controlling hazard causal factors to acceptable levels of safety risk. In many instances, software is called on to compensate for hardware design deficiencies. As software is considered to be cheaper to change than hardware, software design requirements may be derived to control specific hardware causal factors. In other instances, one design requirements (hardware or software) may eliminate or control numerous hazard causal factors. This is extremely important to understand as it illuminates the importance of not accomplishing hardware safety analysis, and software safety analysis separately. A system-level, or subsystem-level hazard can be caused by a single causal factor or a combination of causal factors. The safety analyst must consider all aspects of what causes the hazard and what will be required to eliminate or control the hazard. Hardware, software, and human factors can usually not be segregated from the hazard and cannot be analyzed separately. The analysis performed at this level is integrated into the trade-off studies to allow programmatic and technical risks associated with various system architectures to be determined.

Both software initiated causes and human error causes influenced by software input, must be adequately communicated to the digital systems engineers and software engineers for the purpose of the identification of software design requirements to preclude the initiation of the root hazard identified in the analysis. The software development team should have already been introduced to the applicable generic safety requirements. These requirements should have addressed how the system will react safely to operator errors, component failures, functional software faults, hardware/software interface failures, and data transfer errors. As detailed design progresses, however, functionally derived software requirements will be defined and matured to specifically address causal factors and failure pathways to a hazardous condition or event. Communication with the software design team is paramount to ensure adequate coverage in preliminary design, detailed design, and test.

If we execute a PHL on a system that has progressed past the requirements phase, a list or a tree of identified software-safety-critical functions becomes helpful to flesh out the fault tree, or the tool used to represent the hazards and their causal factors. In fact, the fault tree method is one of the most useful tools in the identification of specific causal factors in both the hardware and software domains.

During the PHA activities, the link from the software casual factors to the system level requirements must be established. If there are causal factors that, when inverted descriptively, cannot be linked to a requirement, they must be reported back to the SSWG for additional

consideration and development and incorporation or additional requirements or implementations into the system level specifications.

The hazards are formally documented to include information regarding the description of the hazard, casual factors, the effects of the hazard, and preliminary design considerations for hazard control by mitigating each cause. Performing the analysis includes assessing hazardous components, safety-related interfaces between subsystems, environmental constraints, operation, test and support activities, emergency procedures, test and support facilities, and safety-related equipment and safeguards. The suggested PHA format (Figure 4-24) is defined by the CDRL and can be included in the Hazard Control Record (HCR) database . This is only a summary of the analysis evidence that needs to be progressively provided to the Safety Data Library to support the residual risk Safety Assessment Report.

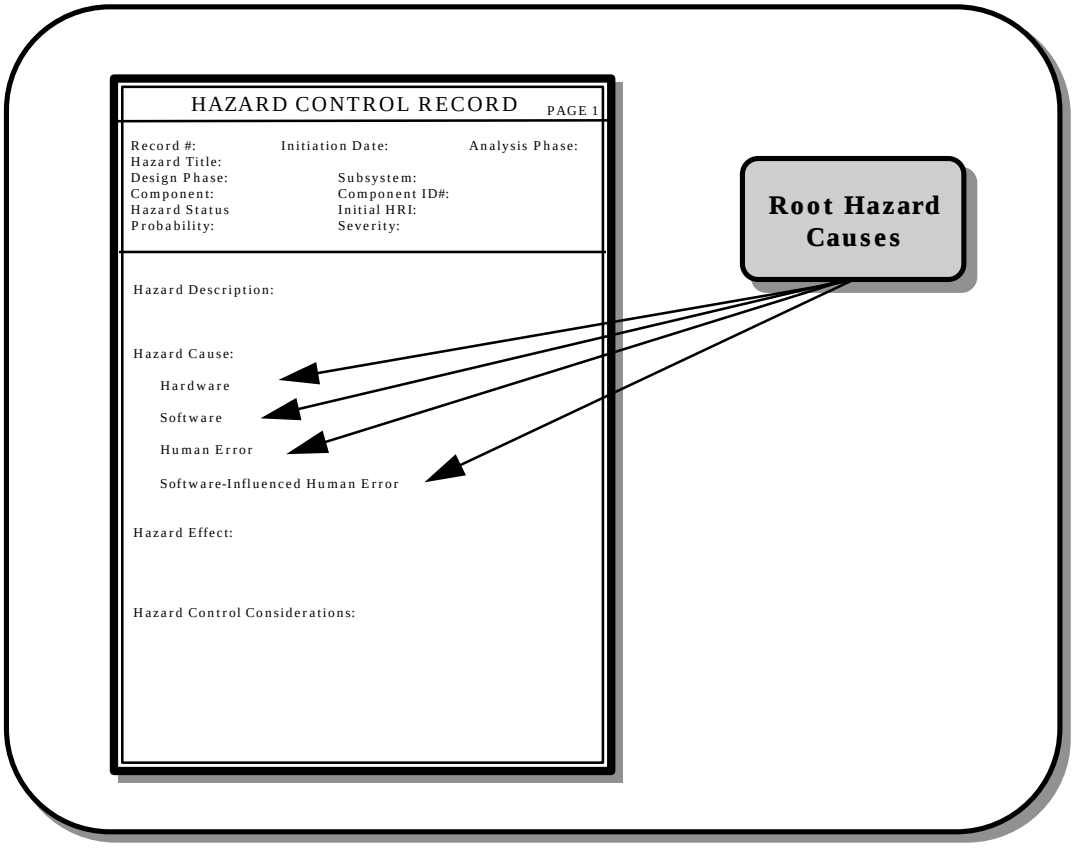


Figure 4-24: PHA Hazard Control Record Example

The PHA document itself is a living document which must be revised and updated throughout the development life cycle. It becomes the input document and information for all other hazard analyses performed on the system. This includes the SSHA and SHA.

4.3.5 DERIVE SYSTEM SAFETY-CRITICAL SOFTWARE REQUIREMENTS

Safety-critical software requirements are derived from known safety-critical functions, tailored generic software safety requirements and inverted hazard causal factors determined from previous

activities. Figure 4-25 identifies the software safety engineering process for deriving system specific, safety-critical software requirements.

Safety requirement specifications identify the specifics and the decisions made, based upon the level of safety risk, desired level of safety assurance, and the visibility of software safety within the developer organization. Methods for doing so are dependent upon the quality, breadth and depth of initial hazard and failure mode analyses and on lessons-learned derived from similar systems. As stated previously, the generic list of requirements and guidelines, establish the beginning point which initiates the system-specific requirements identification process. The identification of system-specific requirements are the direct result of a complete hazard analysis methodology (see Figure 4-26).

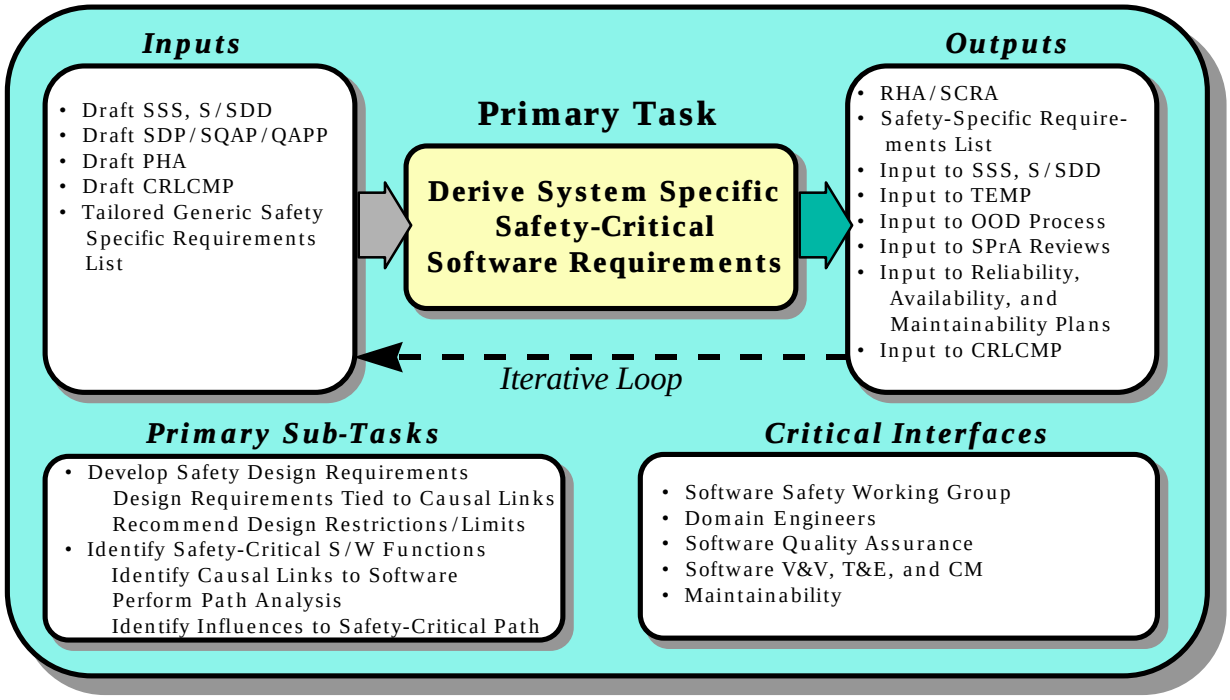


Figure 4-25: Derive Safety-Specific Software Requirements

System-specific software safety requirements require a flow-down of system safety hazards into requirements for the subsystems which provide a trace (audit trail) between the requirement, its associated hazard, and to the module(s) of code that are affected. Once this is achieved as a core set of requirements, design decisions are identified, assessed, implemented, and included in the hazard record database. Relationships to other hazards or requirements are also determined. These derived requirements must be presented to the customer via the SwSWG for concurrence as to whether they eliminate or resolve the hazardous condition to acceptable levels of safety risk

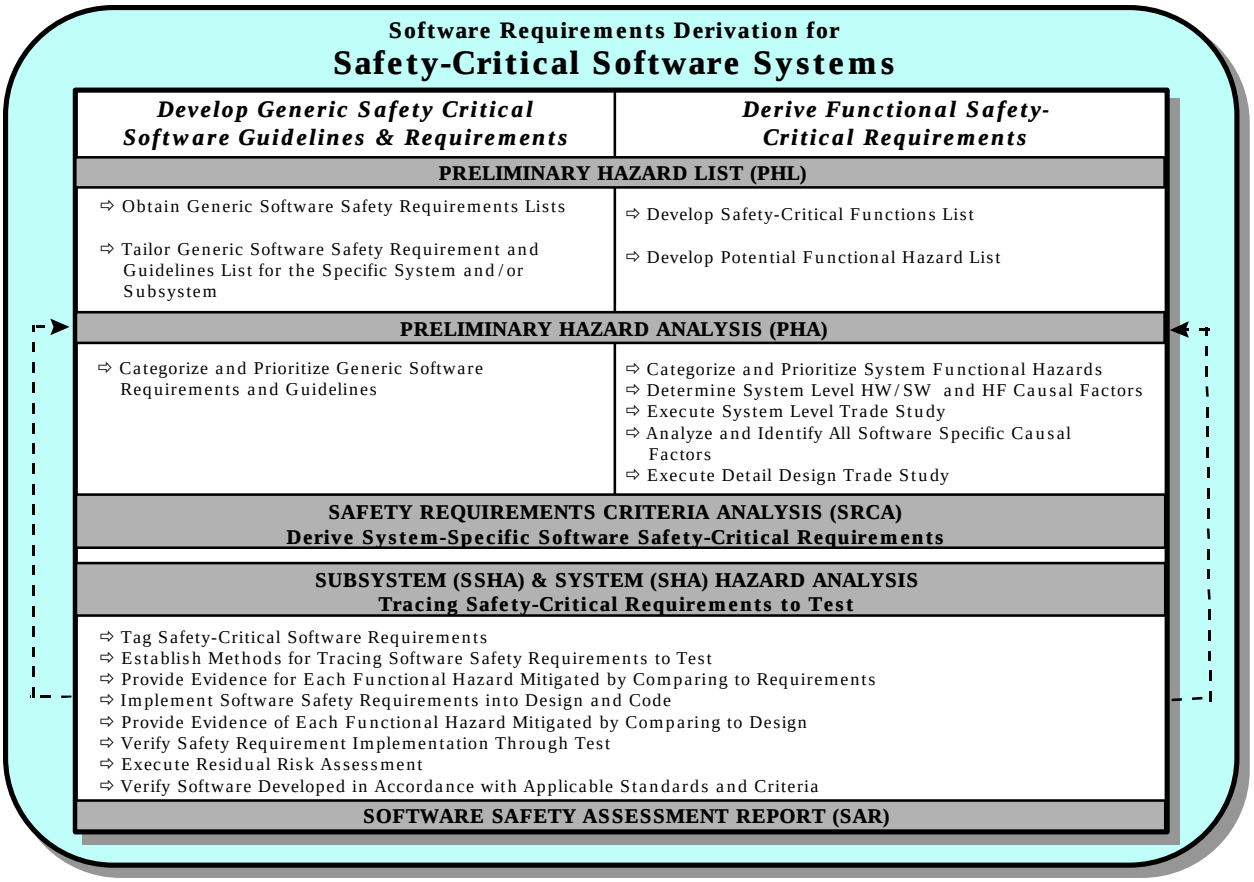


Figure 4-26: Software Safety Requirements Derivation

4.3.5.1 PRELIMINARY SOFTWARE SAFETY REQUIREMENTS

The first "cut" at system-specific software safety requirements are derived from the PHA analyses performed in the early life cycle phase of the development program. As previously discussed, the PHL/PHA hazards are a product of the information reviewed pertaining to systems specifications, lessons learned, analyses from similar systems, common sense, and preliminary design activities. Hazards that are identified during the PHA phase are analyzed and preliminary design considerations are identified to design engineering to mitigate the hazard. These design considerations represent the preliminary safety requirements of the system, subsystems, and their interfaces (if known). These preliminary requirements must be accurately defined in the hazard record database for extraction when reporting of requirements to the design engineering team.

4.3.5.2 MATURED SOFTWARE SAFETY REQUIREMENTS

As the system and subsystem design mature, the requirements unique to each subsystem also matures via the Subsystem Hazard Analysis. The safety engineer, during this life cycle phase of the program, attends the necessary design reviews and spends countless hours with the subsystem designers for the purpose of accurately defining the subsystem hazards. Hazards identified are documented in the hazard database and the hazard "causes" (hardware, software, human error, and software-influenced human error) identified and analyzed. When fault trees are used as the functional hazard analysis methodology, the causal factors leading to the root hazard determine

the derived safety-critical functional requirements. It is at this point in the design that preliminary design considerations are either formalized and defined into specific requirements, or eliminated if they no longer apply with the current design concepts. The maturation of safety requirements is accomplished by analyzing the design architecture to connect the root hazard to the causal factors. The causal factors are analyzed to the lowest level necessary for ease of mitigation (Figure 4-27). The lower into the design the analysis progresses, the more simplistic (usually) and cost effective the mitigation requirements tend to become. The PHA phase of the program should define causes to at least the CSCI level, whereas the SSHA and SHA phases of safety analyses should analyze the causes to the algorithm level where appropriate.

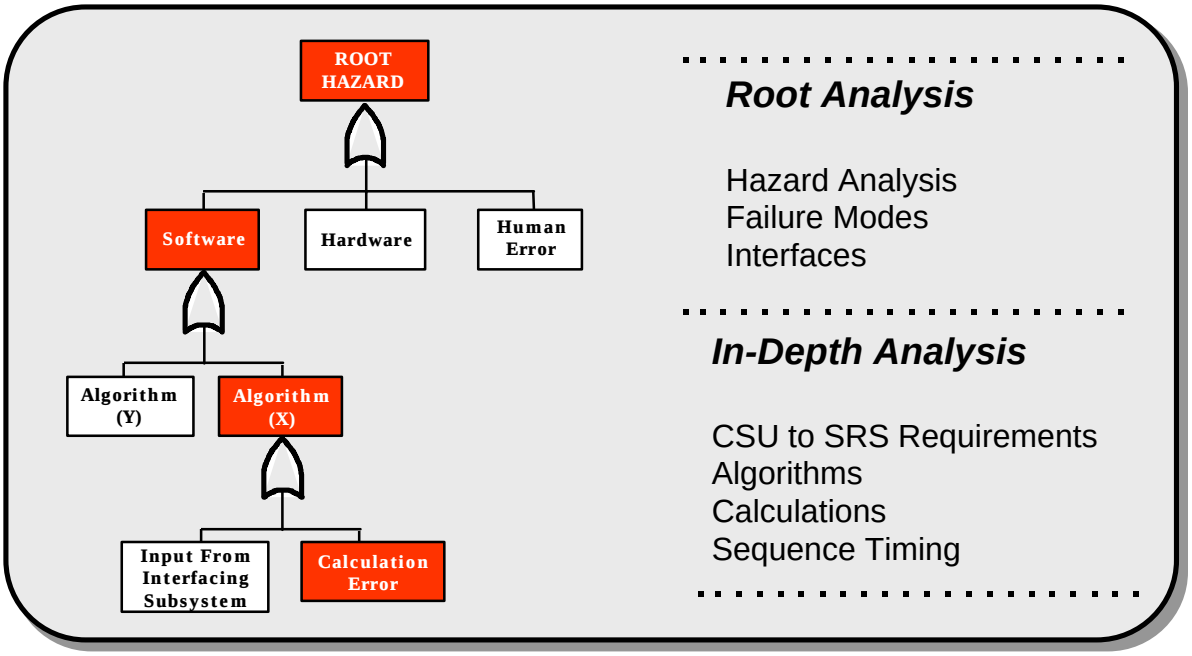


Figure 4-27: In-depth Hazard Cause Analysis

The subsystem analysis begins during concept exploration and continues to be matured through detail design and the critical design review. The safety analyst must ensure that the safety analyses keep pace with the design. As design decisions are made, the affected hazard records must also be reevaluated and updated.

4.3.6 PRELIMINARY SOFTWARE DESIGN, SUBSYSTEM HAZARD ANALYSIS

The accomplishment of a credible software safety program cannot be performed without the identification of subsystem and system hazards and failure modes inherent in the system being developed (Figure 4-28). Today, the primary method of reducing the safety risk of software performing safety-significant functions is to first identify the system hazards and failure modes, and then determine which hazards and failure modes are *caused by* or *influenced by* software or lack of software. This includes scenarios where information produced by software could potentially influence the operator into a wrong decision, or hazardous condition. Moving from "real" hazards to software causal factors and consequently design requirements to eliminate or control the hazard is very practical, logical, and adds utility to the software development process.

It can also be performed in a more timely manner as much of the analysis is accomplished to influence preliminary design activities.

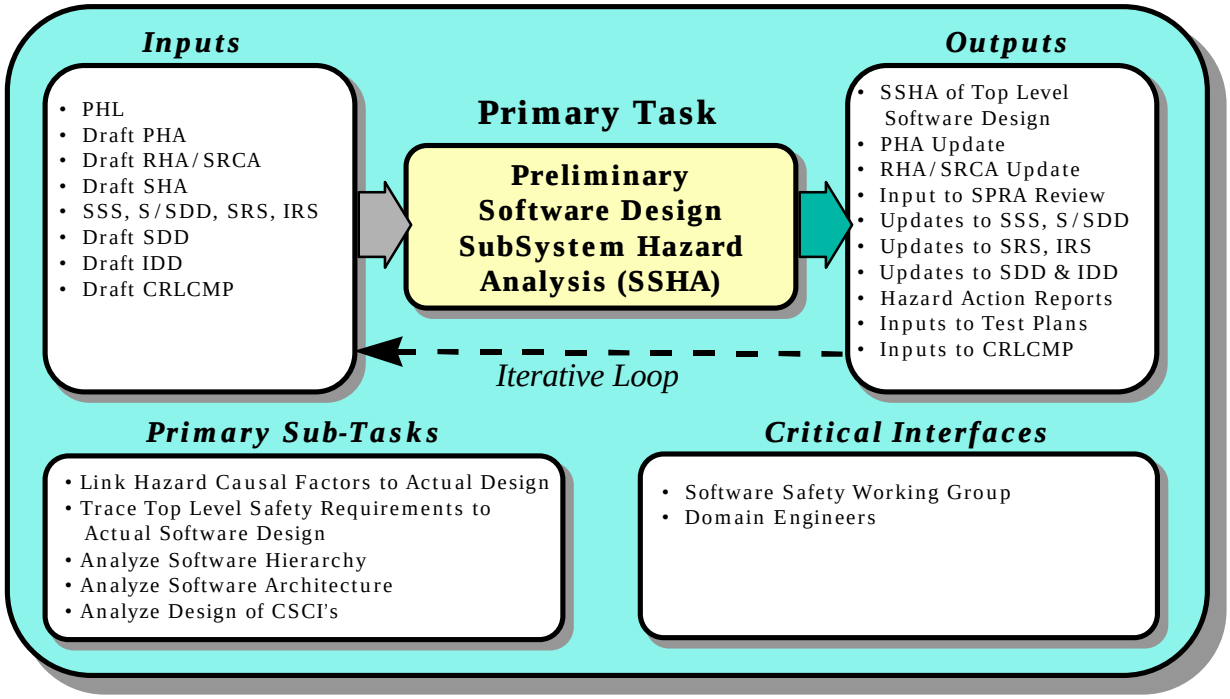


Figure 4-28: Preliminary Software Design Analysis

The specifics of how to perform a subsystem, or system hazard analysis is briefly described in appendices C.1.6 and C.1.7. A more complete description can be found in MIL-STD-882C, Tasks 204 and 205. The fundamental basis and foundation of a system safety (or software safety) program is a systematic and complete hazard analysis process.

One of the most helpful steps within a credible software safety program is to categorize the specific causes of the hazards and software inputs in each of the analyses (PHA, SSHA, SHA, O&SHA, and HHA). Hazard causes can be identified as those caused by; hardware, and/or hardware components; software inputs or lack of software input; human error; and/or software influencing the human error. Hazards can be caused by one specific cause, or any combination of the cause categories. As an example, “loss of thrust” on an aircraft, has the cause potential from all four categories. This could be (1) hardware; blade ingestion, (2) software; software commands engine shutdown in the wrong operational scenario, (3) human error; pilot commands engine shutdown inadvertently, and (4) software influence pilot error; computer provides information to the pilot which is incorrect, pilot makes the wrong decision based on incorrect, insufficient, or incomplete data. On the other hand, some hazards may have only one cause. Whatever the case, hazard control considerations (PHA phase), and requirements (SSHA, SHA, and O&SHA phases) must be identified and defined for the design and development engineers. Hardware causes are communicated to the appropriate hardware design engineers, and software related causes to the software development and design team. All requirements should be reported to the systems engineering group for their understanding and necessary tracking and/or disposition.

The preliminary software design SSHA begins upon the completion of the derivation of the system specific safety-critical software requirements. The purpose is to analyze the system and software architecture and preliminary computer software configuration item design. At this point all generic and functional SSRs should have been identified and it is time to begin allocating them to the identified system level hazards and tracing them to the design.

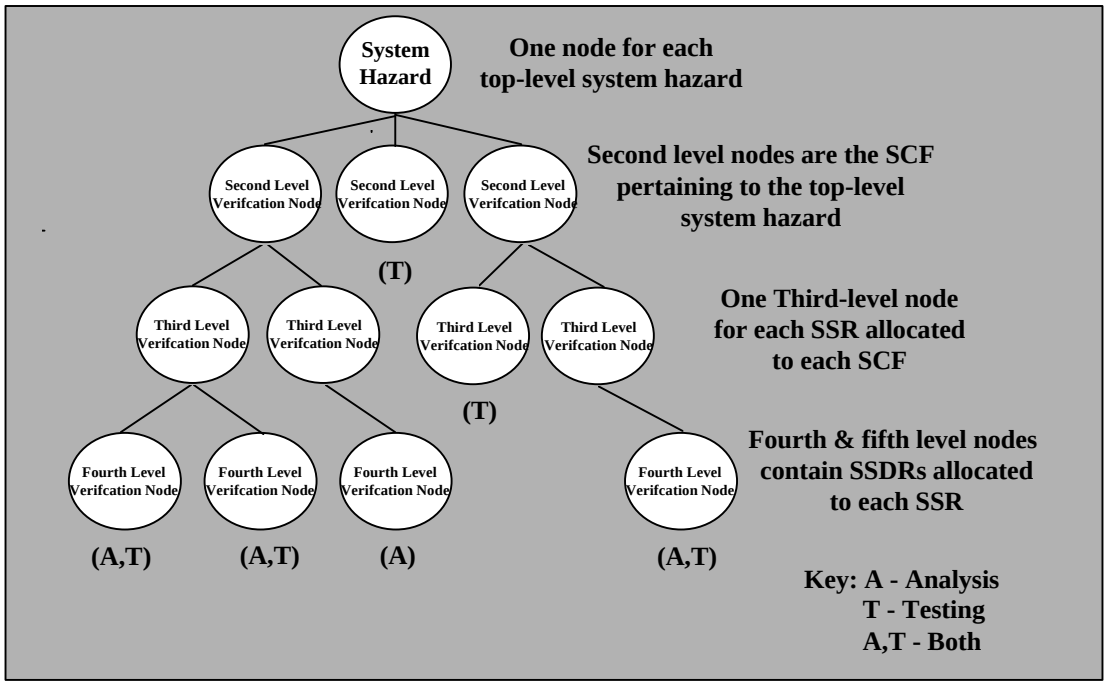


Figure 4-29. SSR Verification Tree

The allocation of the SSRs to the identified hazards can be accomplished through the development of SSR verification trees (Figure 4-29) which links SCFs, SSRs and SSDRs to the top-level system hazards. The tree allows the SSE to verify that controls have been designed into the system to eliminate or control/mitigate the root hazard. The root node of the tree represents one of the system level hazards. A verification tree should be developed for each system level hazard. The second level nodes are the SCFs linked to each system level hazard. The third level nodes represent the SSRs allocated to each SCF. The fourth and fifth level nodes represent the SSDRs that support the SSRs. The fourth and fifth level nodes should be developed as required to fulfill the level of detail required by the SSS Team. By verifying the nodes through analysis, (code/interface, logic, functional flow, algorithm and timing analysis) and/or testing (identification of specific test procedures to verify the requirement), the SSE is essentially verifying that the design requirements have been implemented successfully. The choice of analysis and/or testing to verify the SSR or SSDR is up to the individual SSE and should be based on the criticality of the requirement to the overall safety of the system. When possible, testing should be used for verification. The SSE should also develop an SSR verification matrix, similar to Table 4-2, to track each SSR as it is verified. This SSR matrix should be included as an appendix to the SRCA. The hazard tracking database should be updated with the analysis and test results once the verification has been completed.

SSR	Test/Analysis	Verified (Date)	Comments
1	**Test:** (TP4-1 & TP72-1) **Analysis:** (CSCI/CSU Name)	7/14/97	**Test Passed**. Test Data found on Data Extract Iomega Jaz Disk #10
1.2	**Test: (TP2-2)**	9/1/97	**Test Failed**: STR/PTR JDB002 generated & submitted to design team
1.3	**Analysis:** (CSCI/CSU Name)	9/23/97	Analysis of CSCI/CSU (***Name***) indicated successful implementation of the algorithm identified by SSR 1.3

Table 4-2. Example SSR Verification Matrix

The next step of the preliminary design analysis is to begin tracing the identified SSRs and causal factors to the design (to the actual CSCIs and CSUs). This can be accomplished through traceability analysis. The easiest way to accomplish this task is to develop a Requirements Traceability Matrix as illustrated in Table 4-3. Other methods of PDHA include Module Safety-criticality Analysis and Program Structure Analysis.

SSR	Requirement Description	CSCI	CSU	Test Procedure	Test Results

Table 4-3: Requirements Traceability Matrix Example

4.3.6.1 MODULE SAFETY-CRITICALITY ANALYSIS

The purpose of module (CSCI or CSU) safety-criticality analysis is to determine which CSCIs or CSUs are safety-critical to the system in order to assist the safety engineer in prioritizing the level of analysis to be performed on each module. The priority should be determined by the degree each CSCI or CSU implements a specific safety-critical function or functions.

A matrix (Table 4-4) should be developed to illustrate the relationship each CSCI or CSU has with the safety-critical functions. Include all CSCIs and CSUs that are implicitly required to perform a safety-critical-function, such as math library routines which perform calculations on safety-critical data items. The criticality matrix should list each routine and indicate which safety-critical functions are implemented Symbols could be used to note the importance with respect to accomplishing a safety-critical function.

H - High: The CSCI or CSU is directly involved with a critical factor
M - Medium: The CSCI or CSU is indirectly involved or subordinate to a critical factor
N - None: The CSCI or CSU does not impact a safety-critical function.

The last column in the matrix is the overall criticality rating of the CSCI or CSU. An "H", "M" or "N" should be placed in this column based on the highest level of criticality for that routine.

	Safety-critical Functions						
CSCI/CSU Name	**1**	**2**	**3**	**4**	**5**	**6**	**Rating**
INIT	M		M	M			M
SIGNAL		H	M				H
D1HZ					H		H
CLEAR				H		H	H
BYTE							N

Table: 4-4: Safety-critical Function Matrix

4.3.6.2 PROGRAM STRUCTURE ANALYSIS

The purpose of program structure analysis is to reconstruct the program hierarchy and overlay structure, and to determine if any errors exist in the structure. The program hierarchy should be reconstructed on a CSCI level and its format should be in the form of a control tree. The system safety engineer begins by identifying the highest CSU and its call to other CSUs. This is done for each level of CSUs. When this control flow is completed, the system safety engineer is then able to identify recursive calls, extraneous CSUs, inappropriate levels of calls, discrepancies with the design, calls to system and library CSUs, calls to other CSCIs, overlays, CSUs not called, CSUs called by more than one name, and units with more than one entry point. An example hierarchy tree is illustrated below in Figure 4-30.

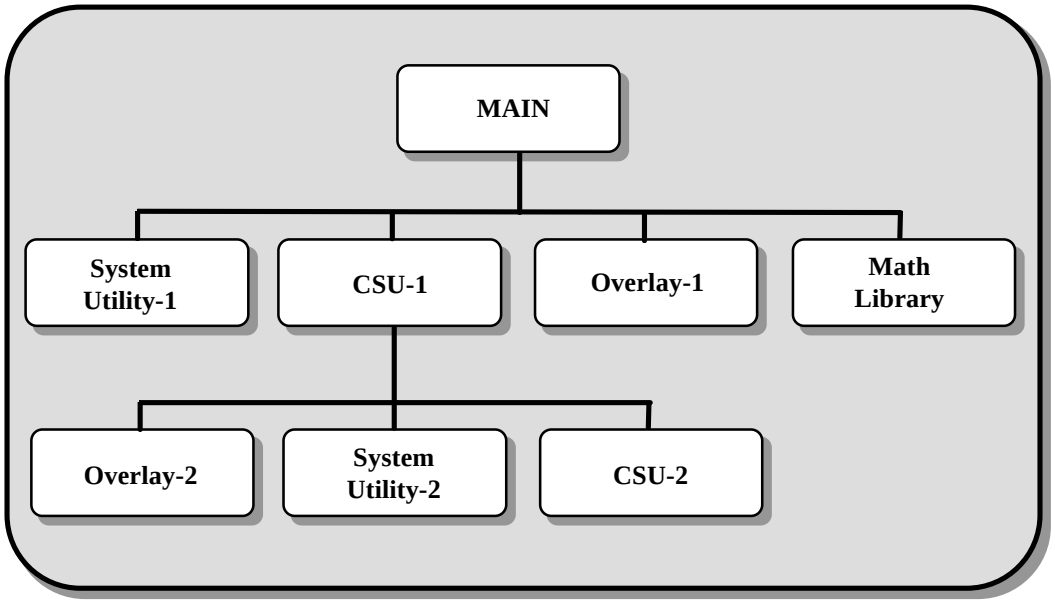


Figure 4-30: Hierarchy Tree Example

All overlays should be identified. After identification, the following issues should be considered:

- Overlaying is not performed unnecessarily (An overlay should not just load another overlay).

- Safety-critical code which should not be in overlays.
- All overlay loads verified and proper actions taken if an overlay cannot be loaded (In some cases the system will halt, in others, some recovery is sufficient, depending on the criticality and impact of the failure).
- The effect of the overlay structure on the time-critical code.
- Interrupts are enabled when an overlay is loaded.
- A review of which overlays are loaded all the time, to determine if they should be made into resident code to cut down on system overhead.

4.3.6.3 TRACEABILITY ANALYSIS

The purpose of traceability analysis is to identify where the safety related requirements are implemented in the code, to identify safety requirements which are not being implemented and to identify code which does not fulfill the requirements. Requirements traced should not just be those identified by the top-level specifications, but those safety related requirements identified by the Software Requirements Specifications (SRS), Software Design Document (SDD) and Interface Control Document (ICD)/Interface Design Specification (IDS). This trace forms the basis for the analysis and test planning by identifying the requirements associated with all of the code. This analysis also ties in nicely with the Safety Requirements Criteria Analysis (SRCA), which not only traces requirements from specifications, to design and test, but helps identify what is safety-critical and what it not.

Tracing encompasses two distinct activities: a requirement to code trace and a code to requirement trace. The forward trace, requirement to code, is performed by first identifying the requirements that belong to the functional area (if they are not already identified through requirements analysis). After identifying the requirements, locate where each requirement is implemented in the code. A requirement may be implemented in more than one place. This can be done one of two ways, either in a matrix form or by directly recording module names next to each requirement in the requirement document.

The backward trace is performed by identifying the code which neither supports a requirement or a “housekeeping” function. In other words the code is extraneous and may be debugging code left over from the software development process. This trace is performed through an audit of the applicable code after the safety engineer has a good understanding of the corresponding requirements and system processing. Code that is not traceable should be documented. The following items should be documented for this activity:

- Requirement-to-code trace
- Unit(s) [code] implementing each requirement
- Requirements that are not implemented
- Requirements that are incompletely implemented
- Code-to-requirement trace

- Unit(s) [code] that are not directly or indirectly traceable to requirements or "housekeeping" functions.

4.3.7 DETAILED SOFTWARE DESIGN SUBSYSTEM HAZARD ANALYSIS

Detailed design level analysis (Figure 4-31) follows the preliminary design process where the software safety requirements (SSRs) were traced to the CSCI level and is initiated after the completion of the Preliminary Design Review (PDR). Prior to performing this process the safety engineer should have completed the development of the fault trees for all of the top-level hazards, identifying all of the potential causal factors and deriving generic and functional SSRs for each causal factor.

This section will provide the necessary guidance to perform a detailed design analysis at the software architecture level. It is during this process that the system safety engineer works closely with the software developers, and IV& V engineers to ensure that the SSRs have been implemented as intended, and to ensure that their implementation has not introduced any other potential safety concerns.

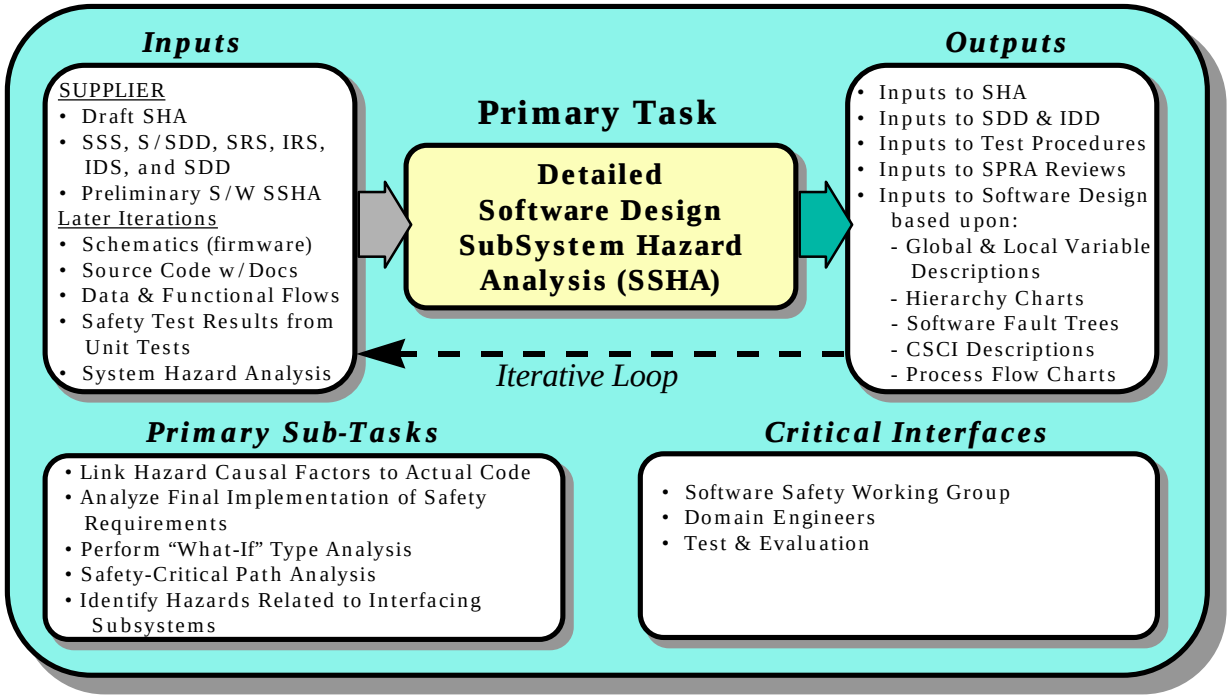


Figure 4-31: Detailed Software Design Analysis

4.3.7.1 PARTICIPATE IN SOFTWARE DESIGN MATURATION

Detailed design analysis provides the software safety engineer (SSE) and the software development experts an opportunity to analyze the implementation of the software safety requirements (SSRs) at the computer software unit (CSU) level. Detailed design analysis takes the software safety engineer from the CSCI level, determined from the preliminary design analysis, one step further into the computer software architecture. As the software development process progresses from preliminary design to detailed design and code, it is the responsibility of

the safety engineer to communicate the SSRs, which are the generic and functional safety requirements derived from the causal factor analysis, to the appropriate engineers and programmers of the software development team. In addition, the safety engineer must monitor the software design process to ensure that software engineers are implementing the requirements into the architectural design concepts. This can only be accomplished by interactive communication between safety engineering and software engineering. It is essential that the software developer and the software safety engineer work together in the analysis and verification of the SSRs. In today's software environment, the software safety engineer cannot be expected to be an expert in all computer languages and architectures. Software design reviews, code walkthroughs, and technical interchange meetings will help to provide a conduit of information flow for the "wellness" assessment of the software development program from a safety perspective. "Wellness" in this situation would include; how well the software design/programming team understands the system hazards and failure modes attributed to software inputs or influences; their willingness to assist in the derivation of safety-specific requirements; their ability to implement the requirements; and their ability to derive test cases and scenarios to verify the resolution of the safety hazard. Most programs today are resource limited. This includes most support functions and disciplines to include system safety engineering. If this is the case, there will not be sufficient time in the day, week, or program for the safety team to attend each and every design meeting. It is the responsibility of the safety

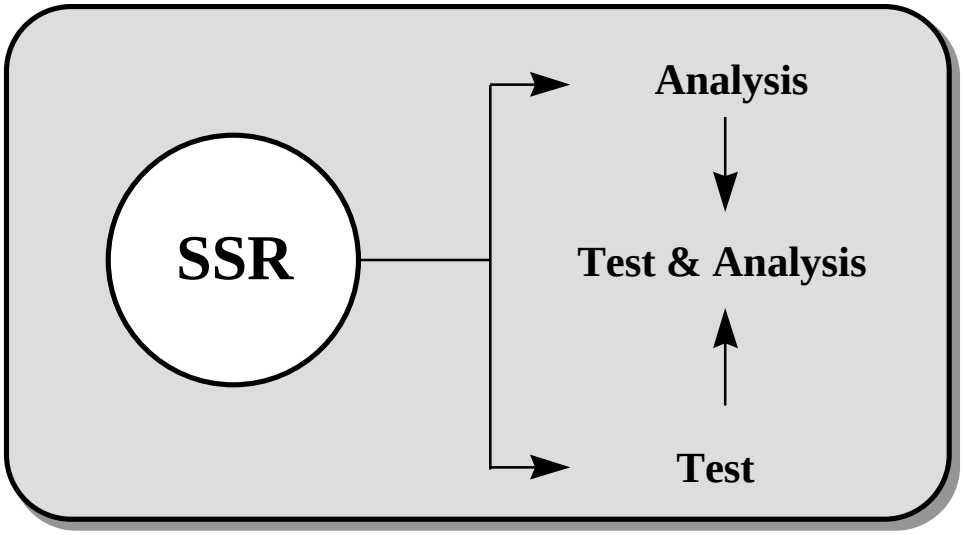


Figure 4-32. Verification Methods

manager to prioritize those meetings and reviews which have the most "value added" to the safety resolution function. This is very dependent on the amount of communication and trust between disciplines, and among team members.

It is important to remember that there is a link between the SSRs and causal factors identified by the fault tree analysis (FTA) during the PHA phase of the software safety process. There are three methods of verifying SSRs: analysis, testing, or both as illustrated in Figure 4-32. Recommended approaches and techniques for analysis will be discussed in the subsequent paragraphs, while approaches for SSR verification through testing will be discussed in Section 4.4.

4.3.7.2 DETAILED DESIGN SOFTWARE SAFETY ANALYSIS

One of the primary analyses performed during detailed design analysis is to identify the Computer Software Unit (CSU) where an SSR is implemented. The term CSU refers to the code-level routine, function or module. The best way to accomplish this task is for the software safety engineer to sit down with the software developer, IV&V engineer or QA engineer and to begin to tag/link individual SSRs to CSUs, as illustrated in Figure 4-33. This accomplishes two goals. First it helps focus the software safety engineer on the safety related processing, which is more important on large scale development projects than on smaller, less complex programs. Secondly, it provides an opportunity to continue development of the Requirements Traceability Matrix (RTM).

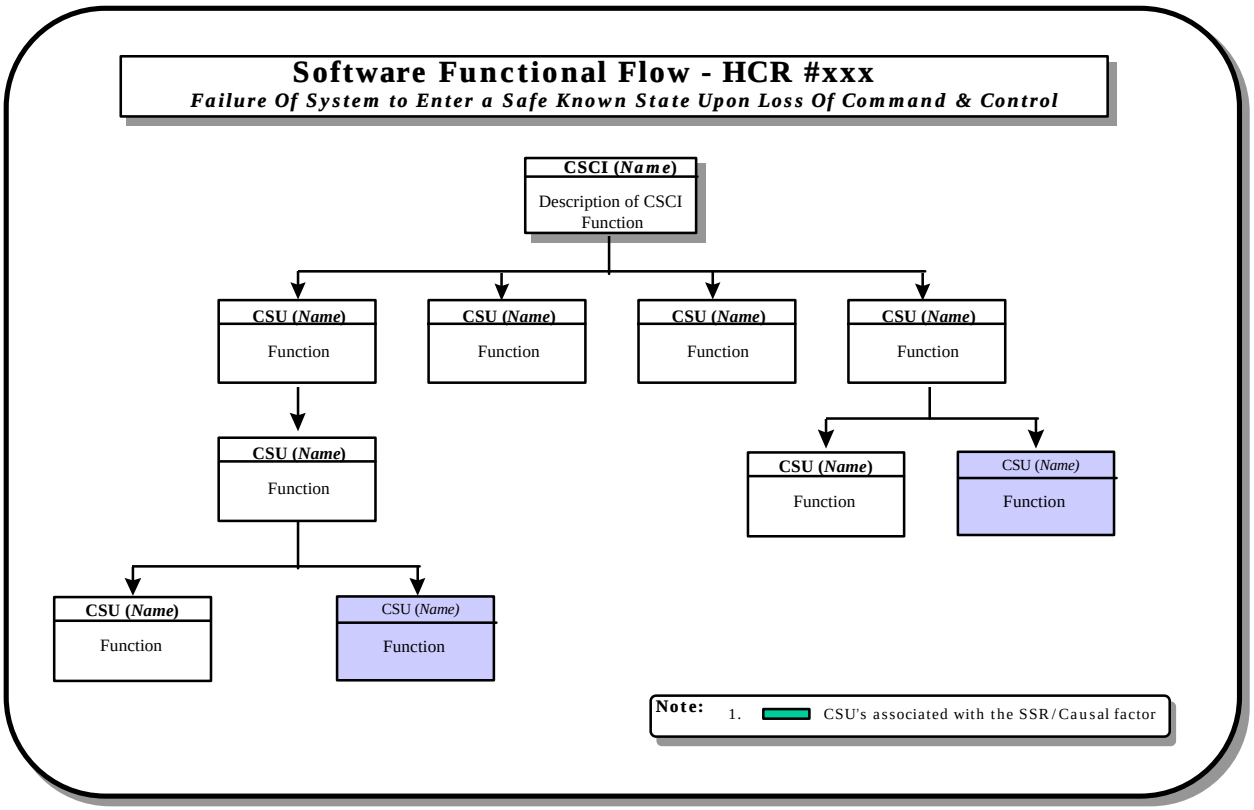


Figure 4-33. Identification of Safety Related CSUs.

The RTM will contain multiple columns, with the left most column containing the list of SSRs. Adjacent to this column will be a description of the SSR, and the name of the CSCI where the individual SSR has been implemented. Adjacent to this column will be a column containing the name of the CSU where the SSR has been implemented. The next column will contain the name of the test procedure that will verify the implementation of the SSR, and the last column is to document test results with comments. As previously discussed, Table 4-3 illustrates the RTM. In some cases it may only be possible to tag/link the SSR to the CSCI level due to the algorithms employed or the implementation of the SSR. If this is the case, the SSR will probably not be verified through analysis, but by an extensive testing effort. The RTM should also be included as part of the Safety Requirement Criteria Analysis (SRCA) report to provide the evidence that all of the safety requirements have been identified and traced to the design and test.

Once the RTM has been populated and all SSRs have been tag/linked to the application code it is time to start analyzing the individual CSUs to ensure that the intent of the SSR has been satisfied. Again, in order to accomplish this task, it is best to have the appropriate developers and/or engineers available for consultation. Process flow charts (PFCs) and data flow diagrams (DFDs) are excellent examples of soft-tools that can aid in this process. These tools can help the engineer review and analyze software safety interlocks such as checks, flags and firewalls that have been implemented in the design. Process flows and DFDs are also useful for performing "What If" types of analyses, performing safety-critical path analyses, and identifying potential hazards related to interfacing systems.

4.3.7.2.1 SAFETY INTERLOCKS

Safety interlocks can either be hardware or software oriented. As an example, a hardware safety interlock would be a keyswitch that controls a Safe/Arm switch. Software interlocks generally require the presence of two or more software signals from independent sources to implement a particular function. Examples of software interlocks are checks and flags, firewalls, come-from-programming techniques and bit combinations.

4.3.7.2.1.1 Checks and Flags

Checks and flags can be analyzed by reviewing the variables utilized by a CSU and ensuring that the variable types are declared and used accurately from CSU to CSU and that they have been logically implemented. As an example lets look at a simple computerized bank checkbook problem containing two CSUs: Debit & Credit. The Debit CSU utilizes a boolean variable as a flag, called "DBT", which it initializes as "1111" or "True" and sets this variable every time the user wishes to make a withdrawal. Meanwhile the Credit CSU utilizes the same flag for making deposits, however it sets it as "1111" or "True" every time the user wishes to make a deposit. This is a simple logic error that could have been the result of two separate programmers not communicating or possibly one programmer making a logic error. This was a simple error and although not life threatening, could cost either the bank or user money simply because the programmer did not utilize unique flags for both the Credit and Debit CSUs. A more life threatening example might be found in a hospital that utilizes computers to administer medication to patients. Within this system is one particular CSU that sets a flag every six hours that signals the machine to administer another dose, however instead of every six hours, due to the flag being implemented within the wrong timer routine (i.e. logic error), the machine sets the flag every hour resulting in an overdose of medication to the patient.

4.3.7.2.1.2 Firewalls

Firewalls are utilized by software developers to isolate one area of software processing from another. Generally they are used by software developers to isolate safety-critical processing from non-safety-critical processing. As an example lets assume that in our medication dosage example there are a series of signals that must be present in order for the medication to be administered. The actual administration of the medication would be considered as safety-critical processing, while the general processing of preparing the series of signals would be non-critical. However, it does take the combination of all of the signals to activate the administration of the medication. The absence of any one of those signals would inhibit the medication from being administered.

Therefore, it would take multiple failures to cause the catastrophic event. In our checks and flags example this type of safety interlock would have prevented the failure of one event causing an overdose.

4.3.7.2.1.3 Come-From Programming

Come-From programming is another example of the implementation of safety interlocks, however it is extremely rigorous to implement and there are very few compilers available on the market that will support this technique. Come-From programming is just another way of protecting or isolating safety-critical code from non-safety-critical code. The difference in this technique is that it requires the application processing to know where it is at all times by using a Program Counter (PC) and to know where it has been (i.e. where it has "Come-From"). By knowing where it is and what the previous processing has been. The application can make validity checks to determine if the processing has stepped outside of its intended bounds. Lets use the medication example again. This time lets require the safety-critical processing CSU, "ADMIN" to only accept an "administer dose" request from CSU "GO". The "ADMIN" CSU would then perform a validity check on the origin of the request. An answer of NOT "GO" would result in a reject and "ADMIN" would either ignore the request, or perform some type of error processing. This type of processing also prevents inadvertent jumps from initiating safety-critical functions. In our example if we suddenly had an inadvertent jump into the "ADMIN" routine, the value of our new PC would be compared to the value of our previous PC. Having preprogrammed ADMIN to only accept the PC from the "GO" CSU, ADMIN would recognize the error and perform the appropriate error processing.

4.3.7.2.1.4 Bit Combinations

Bit combinations are another example of implementing safety interlocks in software. Bit combinations allow the programmer to concatenate two or more variables together to produce one variable. This variable would be the safety-critical variable/signal, that would not be possible without the exact combination of bits present in the two variables that were concatenated together.

4.3.7.2.2 WHAT IF ANALYSIS

"What If" types of analyses are an excellent way to speculate how certain processing will react given a set of conditions. These types of analyses allow the system safety engineer to determine if all possible combinations of events have occurred and to test how these combinations would react under credible and non-credible events. For example, how would the system react to power fluctuation/interrupt in the middle of processing. Would the state of the system be maintained? Would processing restart at the interrupting Program Counter (PC) + 1? Would all variables and data be corrupted? These are questions that need to be asked of code which is performing safety-critical processing to ensure that the programmer has accounted for these types of scenarios and system environments. "What If" analysis should also be performed on all "IF", "CASE" and "CONDITIONAL" statements used in safety-critical code to ensure that all possible combinations and code paths have been accounted for, or to avoid any extraneous or undesired code execution. In addition, this will allow the analyst to verify that there are no fragmented "IF", "CASE" or

"CONDITIONAL" statements and that the code has been programmed top-down and properly structured.

4.3.7.2.3 SAFETY-CRITICAL PATH ANALYSIS

Safety-critical path analysis allows the system safety engineer the opportunity to review and identify all of the possible processing paths within the software and to identify which paths are safety-critical based on the identified system level hazards and the predetermined safety-critical functions of the system. In this case, a path would be defined as a series of events that when performed in serial (one after the other), would cause the software to perform a particular function. Safety-critical path analyses uses the identified system level hazards to determine whether or not a particular function is safety-critical or not safety-critical. Functional Flow Diagrams (FFDs), and Data Flow Diagrams (DFDs) are excellent tools for identifying safety-critical processing paths and functions. In most cases these types of diagrams can be obtained from the software developers or the IV&V team in order to save cost and schedule of redevelopment.

4.3.7.2.4 IDENTIFYING POTENTIAL HAZARDS RELATED TO INTERFACING SYSTEMS

Detailed design analysis also allows the System Safety Engineer an opportunity to identify potential hazards that would be related to interfacing systems. This is accomplished through interface analysis at the Interface Design Specification (IDS)/Interface Control Document (ICD) level. Erroneous safety-critical data transfer between system level interfaces can be a contributing factor (causal factor) to a hazardous event. Interface analysis should include an identification of all safety-critical data variables and ensuring that strong data typing has been implemented for all variables deemed safety-critical. The interface analysis should also include a review of the error processing associated with interface message traffic and the identification of any potential failure modes that would result if the interface fails or the data transferred is erroneous. Failure modes identified should be tied or linked back to the identified system level hazards.

4.3.7.3 DETAILED DESIGN ANALYSIS RELATED SUB-PROCESSES

4.3.7.3.1 PROCESS FLOW DIAGRAM DEVELOPMENT

Process flow diagram development is a line-by-line regeneration of the code into flow chart form. They can be developed by using a standard IBM flow chart template or by free hand drawing. Process flow diagrams provide the link between the functional flow diagrams and the data flow diagrams and allow the system safety engineer to review processing of the entire system in a step by step logical sequence. Process flow diagram development is extremely time consuming and costly, which is one of the reasons it is treated as a related sub-process to Detailed Design Analysis (DDA). If the diagrams can be obtained from the software developers or IV&V team it is an added bonus, but the benefits of reverse engineering the design into process flow chart form does not provide a lot of value to the safety effort in performing hazard causal factor analysis. The real value of process flow diagram development lies in the verification and validation that the system is performing the way that it was designed to perform. The primary benefit to process flow chart development from a system safety viewpoint is that it allows the system safety engineer

an opportunity to really develop a thorough understanding of the system processing, which is essential when performing hazard identification and causal factor analysis. A secondary benefit is that by reverse engineering the coded program into flow chart form, it provides an opportunity for the system safety engineer to verify that all of the software safety interlocks and safety-critical functionality has been implemented correctly and as intended by the top-level design specifications.

4.3.7.3.2 CODE LEVEL ANALYSIS

Code level analysis is generally reserved only for highly safety-critical code due to the time, cost and resources required to conduct the analysis. However, in very small applications, code level analysis may be required to provide adequate assessment of the product. A variety of techniques and tools may be applied to the analysis of code, largely depending on the programming language, criticality of the software, and resources available to the software safety program. The most common method is analysis by inspection. Use of structured analysis methodologies, such as fault tree analysis, Petri Nets, data and control flow analyses, and formal methods is also common at all levels of design and complexity. None of the techniques are comprehensive enough to be applied in every situation, and, are often used together to complement each other.

Data Item Name	Data Type	Data Dimension	Routine Accessing	Global/Local	Common Block
JINCOM	Integer	32 bit	INIT	Global	None
			PROC1 TAB2		

Table 4-5: Data Item Example

Code level analysis always begins with an analysis of the architecture to determine the flow of the program, calls made by the executive routine, the structure of the modules, the logic flow of each module, and finally the implementation in the code. Regardless of the technique used to analyze the code, the analyst must first understand the structure of the software, how it interacts with the system, and how it interacts with other software modules.

The purpose of data structure analysis is to verify the consistency and accuracy of the data items utilized by a particular program. This includes how the data items are defined and that this definition is used consistently throughout the code. One of the best ways to analysis data is to construct a table (Table 4-5) consisting of all of the data items utilized. The table should contain the name of the data item, the data type (Integer, Real, Boolean), the variable dimension (16, 32, 64 bit), the names of routines accessing the data item, whether the data item is local or global, and the name of the common block (if utilized).

4.3.7.3.2.1 Data Flow Analysis

The purpose of data flow analysis is to identify errors in the use of data which is accessed by multiple routines. Except for a very small applications, it would be extremely difficult to

determine the data flow path for every data item. Therefore, it is essential to identify those data items that will affect or control the safety-critical functions of a system from those which do not.

Data flow diagrams (Figure 4-34) should be developed for all safety-critical data items at both the module and system level to illustrate the flow of data. Data is generally passed between modules in one of two ways; globally (common blocks) and locally (parameter passing). Parameter passing is much easier to analyze, since the program explicitly declares which routines are passing data. Data into and out of common blocks should also be traced, but further information will often have to be recorded to understand which subroutines are involved. A table should be developed to aid in the understanding of data flows through common blocks and data passing through several layers of parameters. This table should describe for each variable the subroutine accessing the variable, and how the variable is being used or modified. The table should include all safety-critical variables and any other variables whose use is not clear from the data flow diagram.

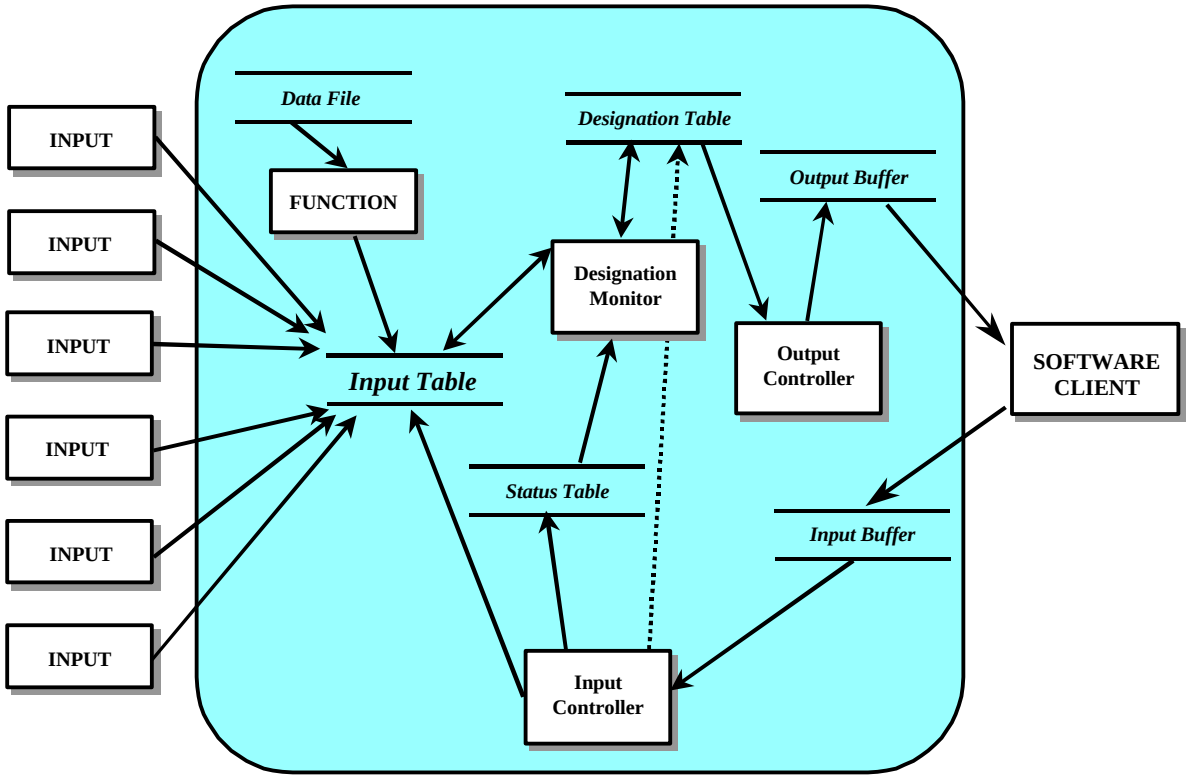


Figure 4-34: Data Flow Diagram Example

An example of errors that can be found from developing both the data item table and the data flow diagrams are:

- Data which is utilized by a system prior to being initialized.
- Data which is utilized by a system prior to being reset.
- Conditions where data is to be reset prior to its use.
- Unused data items.

- Unintended data item modification.

4.3.7.3.2.2 Control Flow Analysis

The purpose of control flow analysis (flow charting) is to reconstruct and examine the logic of the program design language (PDL) and/or code. Constructing flow charts is one of the first analysis activities the analyst can perform as it enables the analyst to become familiar with the code and its design architecture. The draw back to flowcharting is that it is generally costly and time consuming. A better approach is to use the team concept and have the safety engineers interface directly with the software developers and system engineers in order to understand system processing. In most cases the software developers and/or system engineers already have control flow charts developed which can be available for review by the safety engineer as needed. Each case needs to be evaluated to determine which process would be more beneficial and cost effective to the program. In either case flow charting should be done at a conceptual level. Each block of the flow chart should describe a single activity (either single line of code or several lines of code) in a high-level verbal manner, as opposed to simply repeating each line of code verbatim. Examples of both good and bad flow charts can be found in Figure 4-35.

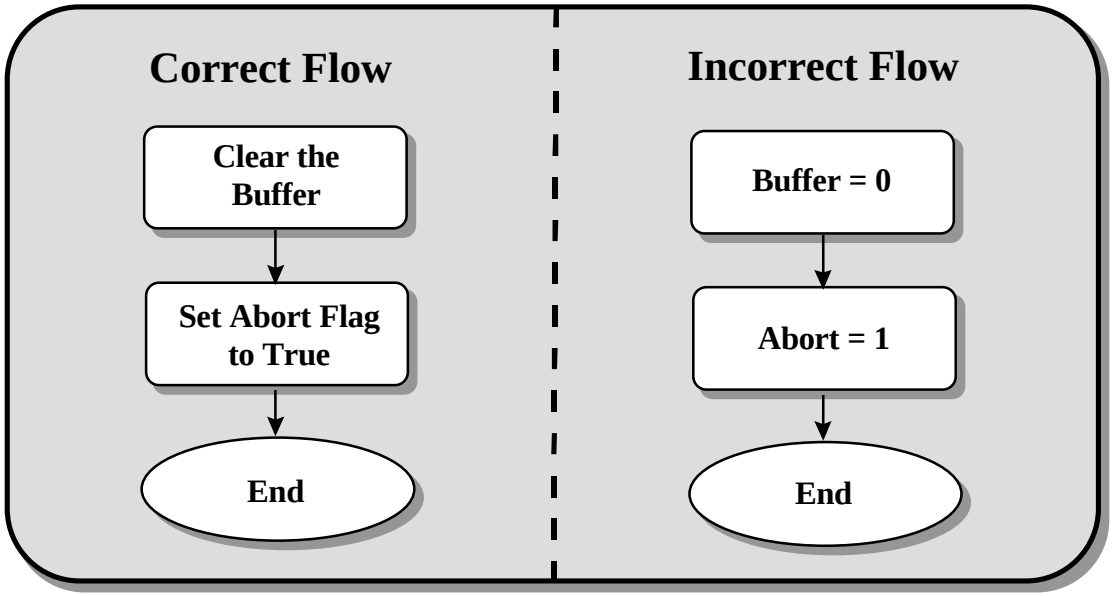


Figure 4-35: Flow Chart Examples

4.3.7.3.2.3 Interface Analysis

The purpose of interface analysis is to verify that the system level interfaces have been encoded in accordance with the IDS/ICD specifications. Interface analysis should verify that safety-critical data transferred between system level interfaces is handled properly. Analyses should be performed to verify how the system functionality will perform if the interface is lost (i.e. casualty mode processing). Analyses should also address timing and interrupt analysis issues in regards to interfaces with safety-critical functions and system components. Performing system level testing and analyzing the data traffic across safety-critical interfaces is generally the best way to verify a

particular interface. Data should be extracted when the system is under heavy stress and low stress conditions to sure the message integrity in maintained in accordance with the IDS/ICD.

4.3.7.3.2.4 Interrupt Analysis

The purpose of interrupt analysis is two-fold. The system safety engineer must first determine the impact of the interrupts on the code, and second determine the impact of the prioritization of the program tasks. Flow charts and program design languages are often used to determine what will happen of an interrupt occurs inside a specific code segment. If interrupts are locked out of a particular segment, the safety engineer must investigate how deep the software architecture will allow the interrupts to be stacked so that none will be lost. If interrupts are not locked out, the safety engineer must determine if data can be corrupted by a low-priority task/process interrupting a high-priority task/process which changes the value of the same data item.

Performing interrupt analysis in a multi-task environment is a little more difficult since it is possible for any task to be interrupted at any given point of execution. It is impossible to analyze the affect of an interrupt on every instruction. In this case, it is necessary to determine segments of code that are tightly linked, such as the setting of several related variables. Interrupt analysis should be limited to these segments in a Multi-Task environment. Items to consider in order to perform interrupt analysis include:

- Program segments in which interrupts are locked out

  Identify the period of time interrupts are locked out.

  Identify the impacts of interrupts being locked out (such as lost messages and lost interrupts)

  Identify possible infinite loops (including loops caused by hardware problems)

- Re-entrant code

  Are sufficient data saved for each activation?

  Is the correct amount of data and system state restored?

  Are units that should be re-entrant implemented as re-entrant?

- Code segments which are interruptible

  Can the interrupted code be continued correctly?

  Will interrupt delay time critical actions (e.g. missile abort signal)?

  Are there any sequences of instructions which should not under any circumstance be interrupted?

- Overall program prioritization

  Are functions such as real-time tasks properly prioritized so that any time critical events will always be assured of execution?

  Is the operator interface of a proper priority to ensure the operator's monitoring

- Undefined interrupts

Are they ignored?

Is any error processing needed?

4.3.7.3.2.5 Analysis By Inspection

Although the most commonly used method of code level analysis, inspection is also the least rigorous. That does not lessen its value to the overall safety assessment process. Analysis by inspection is particularly useful for software that is less critical where a less rigorous methodology is appropriate. Analysis by inspection is also frequently used with other techniques, such as FTAs, control flow analyses, etc., to provide a more thorough assessment of the software. Experience shows that these analysis types are generally complementary, each having strong and weak points. Therefore, they often complement each other to a degree. As noted earlier, a single analysis techniques is usually not totally sufficient to meet the defined safety objectives.

Analysis by inspection is exactly as its name implies - a process whereby the analyst reviews the software source code (high level language, assembly, etc.) to determine if there are any errors or structures that could present a potential problem, either in the execution or in the presence of adverse occurrences, such as inadvertent instruction jumps. To some degree, analysis by inspection relies on heuristics, "clue lists", and engineering judgment.

The ability of the analyst to understand the code as written provides an indication of the ability of future software maintainers to understand it for future modifications, upgrades or correction. Code should be well structured and programmed in a top-down approach. Code that is incomprehensible to the trained analyst will likely be incomprehensible to future software maintainers. The code should not be patched. Patched or modified code provides an opportunity for a high probability of errors since it was rewritten without the benefit of an attendant safety analysis and assessment. The net result is a potentially unsafe program being introduced into a previously "certified" system.

"Clue lists" are simply lists of items that have historically caused problems (such as conditional GO-TO statements), or are likely to be problem areas (such as boundary conditions that are not fully controlled). Clue lists are developed over a long period of time and are generally based on experiences of the analyst, the software development team, or the testing organization. The list below contains several items that have historically been the cause of software problems.

NOTE: <u>It is up to the individual safety engineer to tailor or append to this list based on the language and software architecture being utilized.</u>

- Ensure that all variables are properly defined, and data types are maintained throughout the program.
- Ensure that for maintainability that variables are properly defined and named.
- Ensure that all safety-critical data variables and safety-critical processing are identified.
- Ensure that all code documentation (comments) are accurate and that module headers
- Ensure identified code modifications identified by Trouble Report (TR) and date modification are made.

- Ensure that processing loops have correct starting and stopping criteria (indices or conditions.
- Ensure that array subscripts do not go out of bounds.
- Ensure that variables are correct in procedure call lines (number, type, size, order)
- Ensure that for parameters passed in procedure call lines that Input-Only data is not altered, output data is set correctly, and arrays are handled properly
- Ensure that all mixed modes of operation are necessary, and clearly documented.
- Ensure that self-modifying code does not exist.
- Ensure that there is no extraneous or unexecutable code.
- Ensure local variables in different units do not share the same storage locations.
- Ensure expressions are not nested beyond 5 levels and procedures/modules/subroutine are less than 25 lines of executable code.
- Ensure that all logical expressions are used correctly.
- Ensure that processing control is not transferred into the middle of a loop.
- Ensure that equations are encoded properly in accordance with specifications.
- Exceptions are processed correctly. In particular, if the "else" condition is not processed, will the results be satisfactory?
- Ensure comparisons are made correctly
- Ensure common blocks are declared properly for each routine they are used in.
- Ensure all variables are properly initialized before use.

The thoroughness and effectiveness of an analysis performed by inspection is very dependent on the analyst's expertise, his/her experience with the language and, to a lesser degree, the availability of the above mentioned clue lists. Clue lists can be developed over a period of time and passed on to other analysts. Unfortunately, analysts often tend to keep these lists secret. Many of the design guidelines and requirement of STANAG 4404 are based on such clue lists. However, in this document, the individual clues have been transformed into design requirements (See Appendix C.6).

The language used for software development and the tools available to support that development affect the ability to effectively analyze the program. Some languages, such as Ada and Pascal, force a structured methodology, strong information hiding and variable declaration. However, they introduce complexities and do not support certain functions that are often necessary in system development. Therefore, they are often augmented by other languages. Other languages, such as C and C++, easily support object oriented programming, data input/output structures, and provide substantial flexibility in coding. However, they do not enforce structured programming and modularization, information hiding(in many instances), and the construction of extremely complex program statements that are often incomprehensible even by the programmer.

Hardware, especially the microprocessor or micro-controller, can have a significant influence on the safety of the system irrespective of the computer program. Unique aspects of the hardware may also affect the operation of the machine code in an unexpected manner. Design engineers, especially those often referred to as "bit-fiddlers", take great pride in being able to use unique characteristics of the hardware to increase the efficiency of the code or to make the reading of the machine code as obscure as possible. Occasionally, assemblers also use these unique hardware aspects to increase the efficiency and compactness of the machine code, however, it can pose limitations and possible safety risks.

4.3.8 SYSTEM HAZARD ANALYSIS

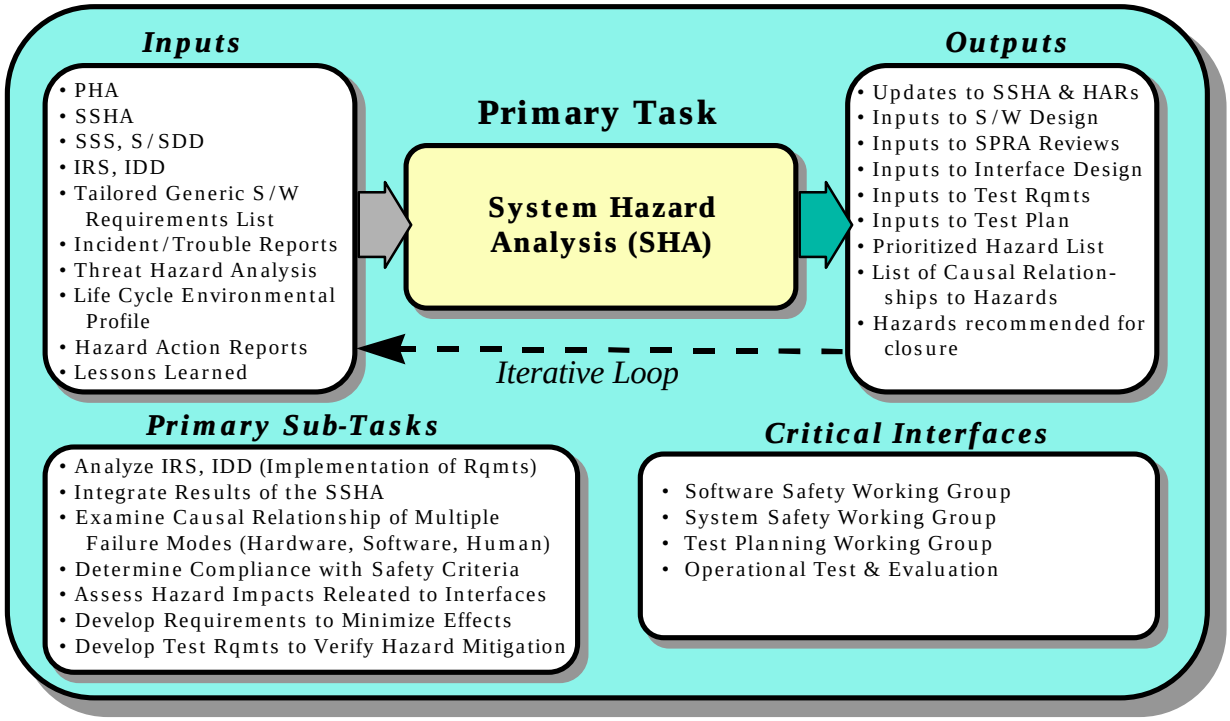


Figure 4-36: System Hazard Analysis

The System Hazard Analysis (SHA) is accomplished in much the same way as the Subsystem Hazard Analysis (SSHA). That is, hazards and hazard causal factors are identified, hazard mitigation requirements communicated to the design engineers for implementation, and the implementation of the safety requirements are verified. However, several differences between the SSHA and SHA are evident. First, the SHA is accomplished during the acquisition life cycle where the hardware and software design architecture is becoming more mature. Second, where the SSHA focused on subsystem-level hazards, the SHA refocuses on system-level hazards which were initially identified by the PHA. In most instances, the SHA activity will identify additional hazards and hazardous conditions because the analyst is assessing a more mature design than that which was assessed during the PHA activity. And third, the SHA activity will put primary emphasis on the physical and functional interfaces between subsystems, operational scenarios, and human interfaces.

Figure 4-36 graphically represents the primary sub-tasks associated with the SHA activity. Due to the rapid maturation of system design, the analysis performed at this time must be in-depth, and as timely as possible for the incorporation of any safety requirements derived to eliminate or control the system-level hazards. As with the PHA and the SSHA, the SHA must consider all possibilities of causes to these hazards. This includes hardware causes, software causes, human error causes, and software-influenced human error causes. The activity of analyzing hazard causal factors to the level, or depth, necessary to derive mitigation requirements will aid in the identification of physical, functional, and zonal interfaces. In a majority of the hazards, the in-depth causal factor analysis will identify failure modes (or causal factor pathways) which will cross physical subsystem interfaces, functional subsystem interfaces, and even contractor/subcontractor interfaces. This is graphically depicted in Figure 4-41 using a simplistic fault tree as an example. In this example, the analyst uses a fault tree approach to analyze a system-level of hazard "Loss of Thrust Actuation". This hazard is depicted as the top event of the fault tree. The SHA activity analyzes all causes to the hazard to include the software branch which is a branch of the "OR" gate to the top-level event. Although this hazard would possess hardware causes (actuator control arm failure), or human error causes (pilot commands shut-down of control unit), the software contribution to the hazard will be the branch discussed.

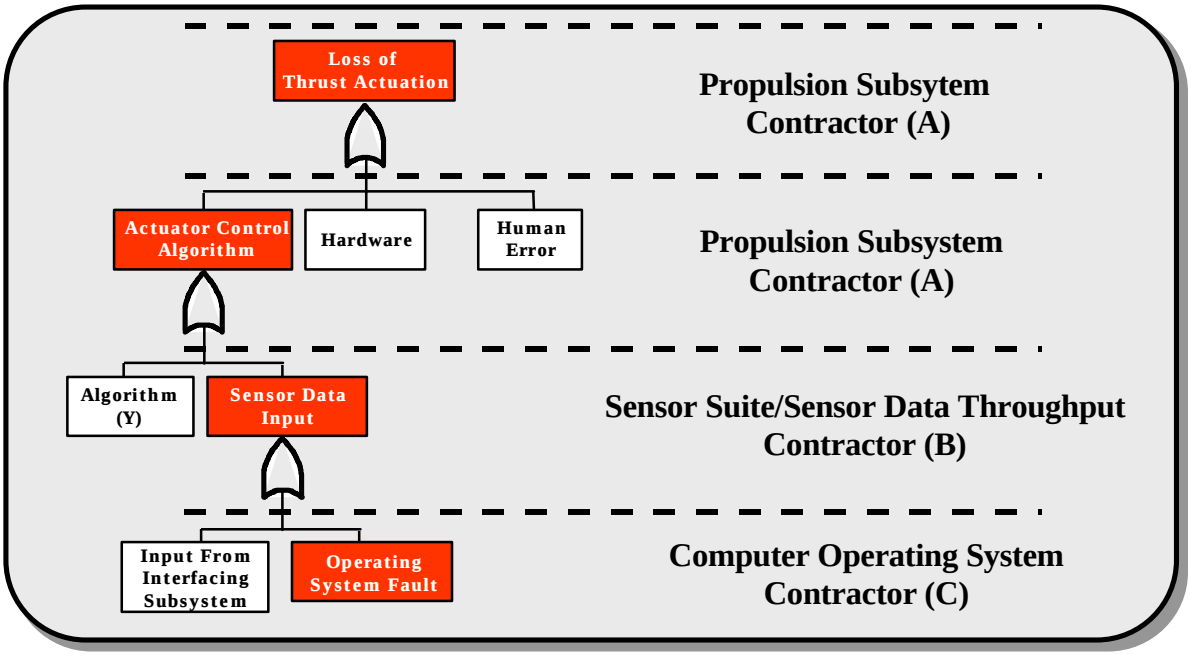


Figure 4-37: SHA Interface Analysis Example

In this example, "Thrust Actuation" is a function of the propulsion system and administratively controlled by the Propulsion IPT of Contractor "A". The computer hardware and software controlling the thrust actuators are also within the engineering boundaries of the same IPT. However, the software safety analyst has determined, in this case, that a fault condition in the computer operating system is the primary causal factor of this failure mode. This operating system fault did not allow actuator sensor data to be read into sensor limit tables and allowed an overwrite to occur in the table. This sensor data was being utilized by the actuator control algorithm. In turn, the actuator control CSC functional architecture could not compensate for

loss of credible sensor data which transitioned the system to the hazardous condition. In this example, the actuator and controlling software is designed by Contractor A, the Sensor Suite and Throughput Data Buss is designed by Contractor B, and the Computer Operating System developed by Contractor C.

Demonstrated in this example, is the safety analysis performed by Contractor C. If Contractor C is contractually obligated to perform a safety analysis (and specifically a software safety analysis) on the computer operating system, the ability to bridge (Bottom Up Analysis)from a operating system software fault to a hazardous event in the propulsion system is extremely difficult. The analysis may identify the potential fault condition, but not identify its system-level effects. The analysis methodology must rely on the "clients", or Contractor A, of the software operating system to perform the Top-Down analysis for the determination of causal factors at the lowest level of granularity.

In-depth causal factor analysis during the SHA activities will provide a spring-board into the functional interface analysis required at this phase of the acquisition life cycle. In addition, the physical and zonal (if appropriate) interfaces must be addressed. Within the software safety activities, this deals primarily with the computer hardware, data busses, memory, and data throughput. The safety analyst must ensure that the hardware and software design architecture is in compliance with the design specifications criteria. As the preceding paragraphs pertaining to PHA and the SSHA (preliminary and detailed code analysis) addressed analysis techniques, they will not be presented here. This section will focus primarily on the maturation of the hazard analysis and the evidence audit trail to prove the successful mitigation of system, subsystem, and interface hazards.

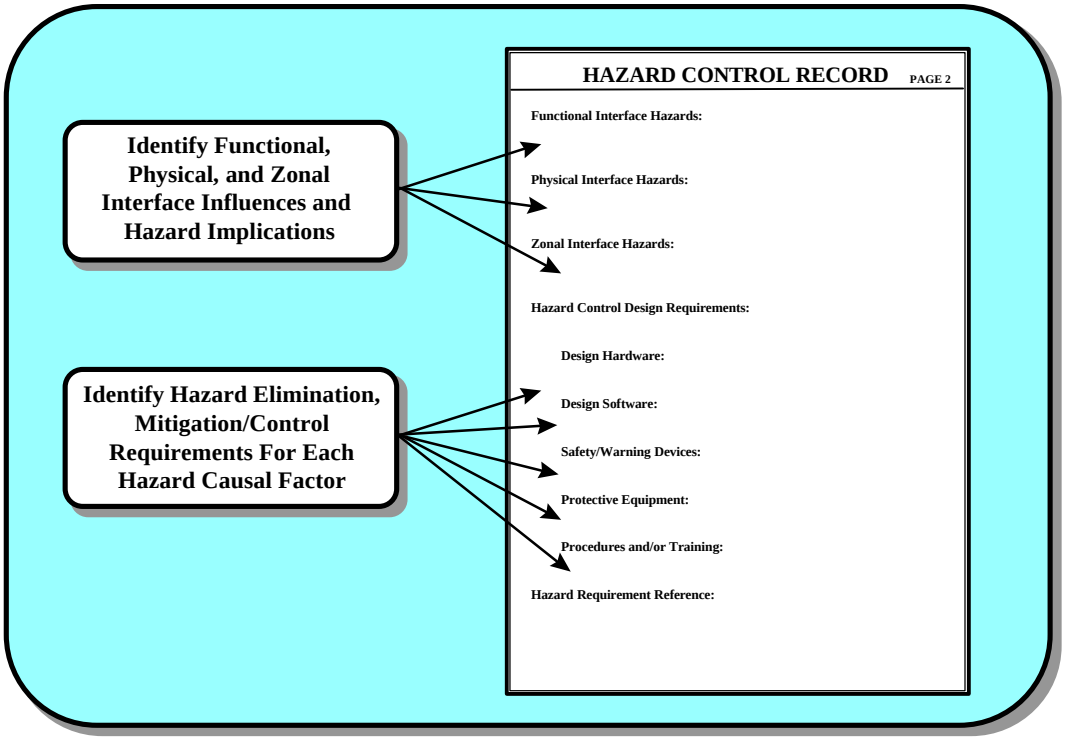


Figure 4-38: Documentation of Interface Hazards and Safety Requirements

Hazard causal factor analysis and the derivation of safety-specific hazard mitigation requirements have been discussed previously in terms of the PHA and SSHA development. In addition, these sections demonstrated a method of documenting all analysis activity in hazard tracking database to provide the evidence of hazard identification, mitigation, and residual risk. Figure 4-29 (of the PHA) specifically depicted the documentation of hazard causes in terms of hardware, software, human error, and software-influenced human error. As the system design architecture matures, each safety requirement that helps either eliminate or control the hazard must be formally documented (Figure 4-38) and communicated to the design engineers. In addition, the SHA activities must also formally document the results of the interface hazard analysis.

At this point, the safety analyst must focus on the ability to define safety test and verification requirements. The primary purpose of this activity is to provide the evidence that all safety requirements identified for hazard elimination or control have been successfully implemented in the system design. It is quite possible that the analyst will discover that some requirements have been implemented in total, others partially, and a few which were not implemented. This is why active involvement in the design, code, and test activities is paramount to the success of the safety effort.

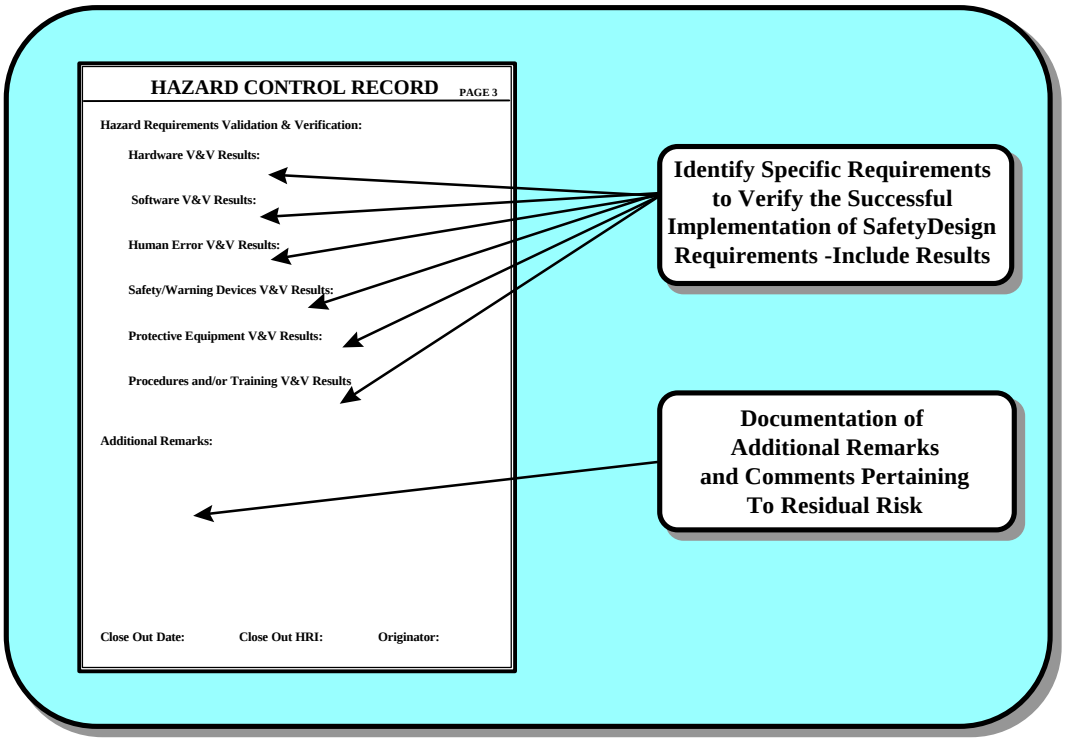


Figure 4-39: Documenting Evidence of Hazard Mitigation

The ability to assess the system design compliance to specific safety criteria is predicated on the ability to verify safety requirements through test activities. Figure 4-39 depicts the information required in the hazard control records of the database to provide the evidence trail required for the risk assessment activities.

4.4 SOFTWARE SAFETY TESTING & RISK ASSESSMENT

4.4.1 SOFTWARE SAFETY TEST PLANNING

Software testing is an integral part of any software development effort. Testing should not only address those performance related requirements, but those that are safety related. The SSS Team must interface directly with the software developers and IV&V team to ensure that all software safety requirements, as identified by the SSS Team to the software developers, have been implemented and that the system functions in accordance with the design specifications.

The software safety testing effort should be highly integrated with the IV&V team, in order to save time and cost. The software safety engineers, through the SSS Team, can identify a large amount of safety related requirements that can be tested and verified by the IV&V team. Those safety related requirements that require testing can either be tested independently by the SSS Team, or added to the IV&V team. The later is recommended.

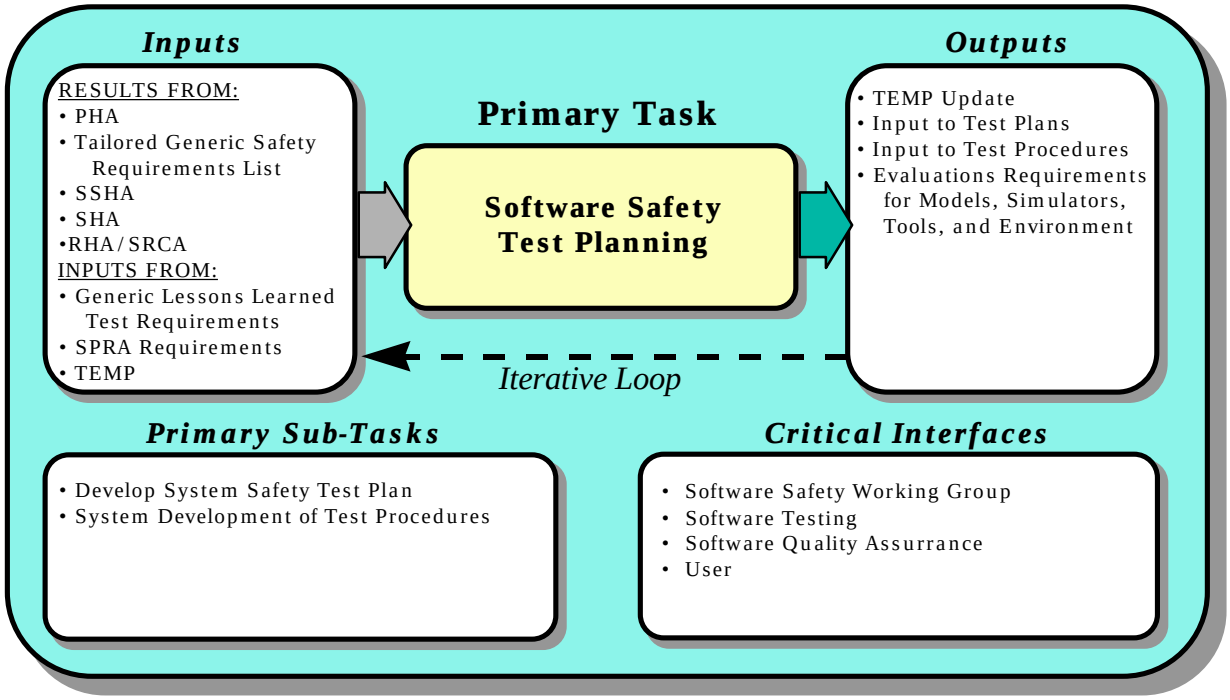


Figure 4-40: Software Safety Test Planning

The software test planning process must address all of the simulators, models, emulators, and software tools that will be utilized by the test team (either IV&V or safety) in order to ensure that all processes, requirements and procedures for validation are in place. It is also important that the software safety test plan address how the software safety engineer will be an active participant in all Test Planning Working Group (TPWG) meetings and how inputs will be provided to the Test Readiness Review and Safety Program Review Authority.

Outputs from this software safety test planning process include an updated IV&V Plan, an updated TEMP, evaluation requirements for simulators, models, emulators, and test environments and tools and an updated Software Test Plan (STP).

The software safety test planning activity must be integrated into the overall software testing plan and the TEMP. Safety-critical code should be identified as well as all safety related requirements. The SRCA (Appendix C.1.8) must provide all of the safety related requirements necessary for inclusion . The software safety test planning must also address the schedule for testing (Functional Qualification Testing [FQT] and system-level testing) all of the software related requirements. This schedule will need to be integrated into the overall software test schedule and TEMP. The software safety test schedule will be largely dependent on software test schedule. Beware of schedule compression due to late software development. Allow time for an effective analysis of test results. It is also important that the SSS Team identify several system level test procedures or benchmark type tests that will verify that the hazards identified during the hazard analyses (SHA, SSHA) have been either eliminated, mitigated, or controlled.

During the software safety test planning process it is necessary to update the Requirements Traceability Matrix (RTM) developed during the SRCA. At this point requirements are linked to actual test procedures. All safety related requirements should be linked to at least one IV&V Test Procedure. There will be many requirements that will be linked to multiple test procedures. By linking safety related requirements to test procedures the safety engineer will be able to verify that all safety related software will be tested. If requirements exist which cannot be linked to existing test procedures, the safety engineer can either recommend that the SSS Team develop a separate test procedure to be executed independently, or recommend that an additional test procedure be added to the IV& V test procedures.

It is important that the SSS Team review all of the IV&V test procedures that have safety related requirements linked to them to ensure that the intent of the safety requirement will be satisfied by the test procedure. This also holds true for the development of new test procedures. If a specific test procedure fails to address the intent of a requirement it is the responsibility of the SSS Team to recommend the appropriate modifications to the IV&V team during TPWG meetings.

4.4.2 SOFTWARE SAFETY TEST ANALYSIS

Software testing (Figure 4-41) is generally grouped into three levels: unit testing, integration testing, and system integration testing. However, within each level there are often numerous sub-levels. This is especially true in integration testing. In addition to the software under development, the software engineering IPT may develop support software to include simulation, emulation, stimulation, run-time environment, data extraction and reduction software, and math models. The SSS Team must participate in the specification of these programs, just as they do for the application software under development. They will need to specify capabilities of the simulators and stimulators, such as the ability to induce faults into the system and to test its response to those conditions. The SSS Team must also specify the parameters to be extracted to perform the necessary safety assessment. To a large extent, this process needs to occur up front to permit the software engineering team adequate time to develop these programs.

At the unit level, testing is generally limited to the functionality of the unit or module. However, to test that functionality, the developer requires test drivers and receivers to either stimulate the unit under test or simulate programs with which the unit or module communicates. Integration testing is performed at several levels beginning with building modules from units and gradually progressing up to system level testing. Integration testing begins with interfacing units that have

completed unit level testing to ensure that they interact properly. Additional units are added until a configuration item is completely tested. Several sets of integration testing may occur in parallel in complex systems involving several CSCIs. CSCIs are then integrated at the next level until the complete software system is tested. Finally, the total system, hardware and software, is subjected to integration testing at the system level with simulators, stimulators, and run-time environments. However, regardless of the degree of sophistication in the testing, laboratory testing is limited by the fact that the environment is very different from the actual environment that the system will be deployed into. This is caused by the limitations on the simulators which can be developed. It may not be practical nor desirable to implement many requirements due to the inherent complexity and difficulty in validating these programs.

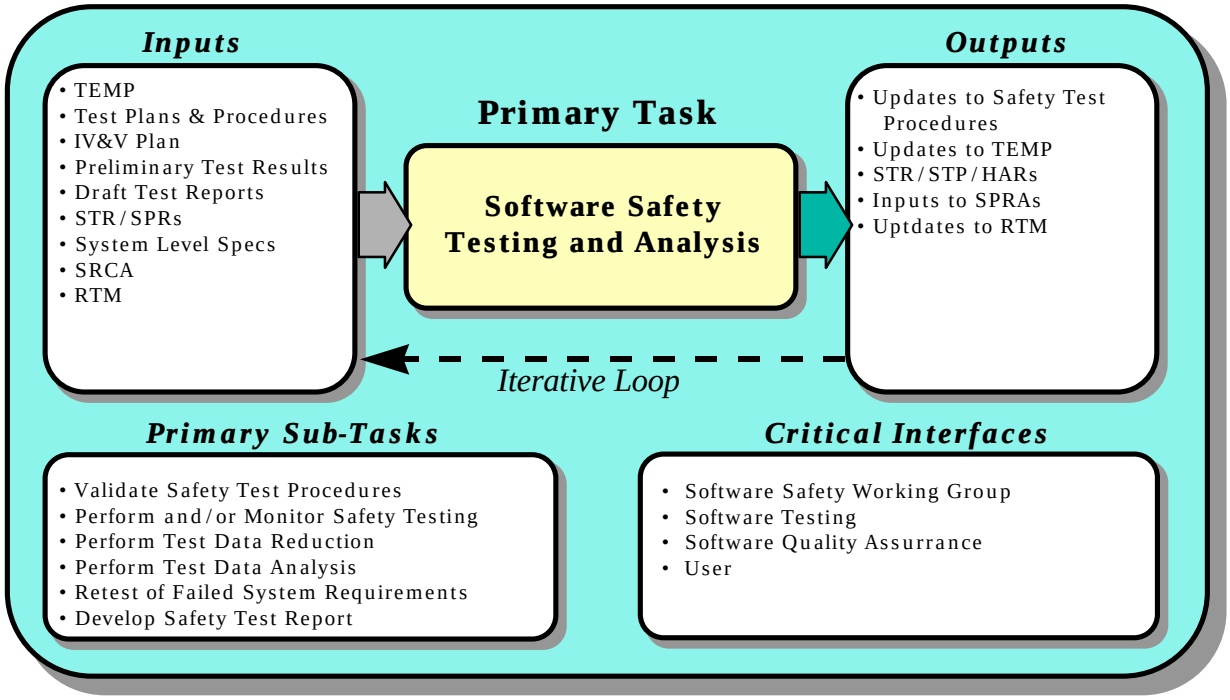


Figure 4-41: Software Safety Test and Analysis

Testing can represent a large portion of the overall software safety effort. The development of safety design requirements, both generic and system specific and the subsequent analysis of their implementation requires that they be verified and validated. Detailed analyses of the design often result in the need to perform tests to verify characteristics, both desirable and undesirable, that cannot be verified through analysis. This is especially true of timing and queuing requirements. The software safety team will identify safety test requirements throughout the system, preliminary and detailed analysis of the system and its related software. Testing begins with the planning phase (see Section 4.4.1) wherein the SSS Team incorporates safety test requirements into program documentation, such as the System Engineering Master Plan (SEMP), the Test and Evaluation Master Plan (TEMP), and the Independent Verification and Validation (IV&V) plans. Generic safety tests, extracted from documents such as STANAG 4404 and lessons learned, provide the basis for early module and integration testing. As analyses proceed, the software safety team identifies additional safety tests and integrates them into appropriate test cases and procedures to ensure full coverage of the safety requirements and safety-critical functions. The

testing phase is complete when the SSS Team complete analysis of test results and assesses the residual risk associated with the software application in the system.

As noted during the requirements analysis phase, the SSS Team must analyze the implementation of the safety design requirements to ensure that the intent of the requirement is met. Likewise, in reviewing the test cases and procedures, the SSS Team must ensure that they will validate the correct implementation of the requirements. "Correct" in this context means that the test cases and procedures verify the intent, not just the letter of the requirement. It differs from the term "correctness" in software engineering which means that the resulting code fulfills the specifications and meets the users' needs. The software safety analyst must ensure that the test environment, test cases, and test procedures will provide the required data to assess safety of the code. Often, the SSS Team can incorporate safety test requirements into routine testing of the software modules, configuration items, and at the system level with no impact on the testing schedule or process.

Inherent to the software safety testing process is the need for the SSS Team to provide as much information as possible to the software testers, at the module, integration, and system level, regarding the safety-critical aspects of the system and its software. They must identify those portions of the code that are safety-critical, at both a module level and at a functional level. They must also identify those tests that are safety specific or those portions of other tests that have a bearing on the safety aspects of the system. The most effective method of identifying the safety-critical code is to establish early in the system development process a means of marking it in the system documentation.

As part of the safety test cases, the SSS Team must provide testers with a list of the test environment conditions (e.g., simulators, drivers, etc.), test procedures, expected and undesired outcomes, and the data extraction requirements. Often, the testing organization will assume responsibility for this development if the SSS Team provides them the necessary guidance and training. Although this may seem prohibitive from a time standpoint, the software testers possess the detailed knowledge and experience necessary to design the most effective, efficient tests to help streamline the effort. However, even if the testers assumes this responsibility, the SSS Team must review the test procedures and the test results to ensure their completeness and accuracy.

Test procedure validation requires that the analyst examine the procedures being establish and verify that they will completely test the safety design requirement or verify that potential failure modes or causal factors cannot result in a hazardous condition. The SSS Team must monitor the safety testing to both validate the procedures and make any adjustments necessary to spot anomalies that may have safety implications. This does not mean that a member of the SSS Team be present for every test run. Often, the responsibility can be assigned to a member of the test team providing they have the appropriate guidance and training. The software test team must annotate anomalies noted during the tests on the Software Trouble Report (STR), as they relate to test procedures or test outcomes. With the proper training and guidance, they will be able to identify potential safety related anomalies.

A majority of the anomalies will not be uncovered until the testing organization reduces the test data into a usable form. Data reduction involves extracting the data recorded during the tests, processing the data such that performance and other parameters can be derived, and presenting the information in a readable, understandable format. For example, in testing an operational flight

program (OFP) for an aircraft, the displays and data presented to the testers may appear valid. However, until that data is compared to that generated by a math model, the results are uncertain. Upon comparison, the testing organization may discover anomalies in the data (e.g., altimeter readout: errors of a few feet during landing can be disastrous yet appear acceptable in the laboratory). At this point, the software development organization must determine whether the anomalies are a result of errors in test procedures, the test environment, the OFP, or in the math model itself. If the errors are in the procedures or environment, the testing organization can make the appropriate changes and re-run the test. If the errors are in either the OFP or the math model, the analysts must determine what the anomaly is and the necessary correction. In either case, the proposed changes must be approved by the configuration management team and submitted through the appropriate configuration control process, including the analysis and testing regimen, before being implemented. The software testing organization performs regression testing and then repeats the test case(s) that failed. Correction of errors often unmask errors or introduce new ones. One study found that for every two errors found and corrected, one was either unmasked or a new one introduced. Therefore, regression testing is an essential part of the overall testing process. It ensures that the modifications made do not adversely affect any other functionality or safety characteristics of the software. The SSS Team must participate in the development of the regression test series, at all levels, to ensure that safety requirements are revalidated for each change.

Obviously, in a complex system, the above described process will result in a substantial number of STR's and modifications to the software. Often, the software configuration control board will group several STR's affecting a single module or configuration item together and develop a one-time fix. As discussed in Section 4.5.1, the SSS Team must be in the STR review process to ensure that the modifications do not adversely affect the safety of the software and to permit updating analyses as required.

In reviewing extracted data, the SSS Team must know those parameters that are safety-critical, and the values that may indicate a potential safety problem. Therefore, in the development of the data extraction, the software testers and SSS Team must work closely to ensure that the necessary data be extracted and that the data reduction programs will provide the necessary data in a readable format for use by the software safety and software testing groups. Often, this requires interaction with domain experts to provide the necessary detailed knowledge of significant parameters. If the data extraction and reduction programs are written correctly with sufficient input from the SSS Team, the resulting printout will identify those parameters or combinations of parameters that represent a potentially hazardous condition. However, there is no substitute for analysis of the data since not all potentially hazardous conditions can be identified in advance.

Part of the data extraction may be a needed to monitor conditions that cause the software to execute specific paths through the program (such as no-go paths). Often, these parameters are difficult or impossible to extract unless the program contains code specially designed to output that data (write statements, test stubs, breaks) or unless the test environment allows the test group to obtain a snapshot of the computer state (generally limited to monitoring computer registers). Unfortunately, test stubs often change the conditions in the executing software and may result in other errors being created or masked. Therefore, their use should be minimized. If used, the software testing organization must perform regression testing after these stubs have been removed

to ensure that errors have not been masked, additional errors introduced by their removal, or timing errors created by their removal.

The purpose of data extraction and analysis is to identify safety related anomalies and subsequently the causal factors. The causal factors may be errors in the code, design, implementation, test cases, procedures and/or test environment. If the casual factor cannot be identified as a test procedural error, test case error, or test environment error, the software safety analyst must analyze the source code to identify the root causes of the anomaly. This is the most common reason for performing code level analysis. As described in Section 4.3.7, once the causal factor is identified, the analyst develops a recommended correction, in coordination with the software developer, and presents it via the STR process to the software configuration control board. The analyst should also determine whether specific generic guidelines and requirements have been adhered to.

Occasionally, the software safety analyst will identify additional safety requirements that must be incorporated into the system or software design. These are submitted as recommended corrective actions through the STR process. The SSS Team must perform a preliminary assessment of the risk associated with the new requirement and present it as part of a trade-off study to the systems engineering IPT. The SSS Team will write up the recommended addition as an Engineering Change Proposal (ECP), and, if approved, will undergo the same process for analyzing and testing the modification as for an STR.

Software safety requirements cannot always be validated through testing. Often this will show up as a failure in the test for a variety of reasons or a no-test. For example, limitations on the capabilities of simulators, stimulators, or the laboratory environment may preclude passing or completing certain tests. The SSS Team will need to make an engineering judgment as to whether the testing that has been completed is adequate. Additional tests may be required to provide sufficient assurance that the function is safe or to provide the desired level of safety assurance.

Throughout the testing process, the SSS Team will interact closely with both the software testing organization to ensure that safety requirements are addressed at all levels. Together, the groups will assess the results of the testing performed and begin developing the safety test report. The report must identify the tests performed and the results of the analysis of the tests. References to test reports from the software testing group, software trouble reports, engineering change proposals, and anomalies detected and corrected should all be included in the test report. At the conclusion of the testing, the SSS Team uses the results to update the Safety Requirements and Criteria Analysis and the various preliminary and detailed analyses performed on the system and its software. The software safety test report will be appended to the system safety test report and will form a part of the basis for the final system safety assessment report.

4.4.3 SOFTWARE STANDARDS & CRITERIA ASSESSMENT

This subsection provides guidance to the SSS Team to verify that software is developed in accordance with applicable safety related standards and criteria. The assessment (Figure 4-42) begins very early in the development process as design requirements are tailored and implemented into system level and top tier software specifications and continues through the analysis of test

results, and various reports from other IPT's. Ultimately, the assessment becomes an integral part of the overall Safety Assessment Report (SAR).

Standards and criteria include those extracted from the generic documents, such as STANAG 4404, military, federal, and industry standards and handbooks, lessons learned, safety programs on similar systems, internal company documents, and other sources. Verification that the software is developed in accordance with applicable software engineering standards and criteria and syntactic restrictions is largely a function that the SSS Team can delegate to the Software Quality Assurance, Software Configuration Management (SCM) and Software Testing (including the IV&V) teams. This is especially true with many of the generic software engineering requirements from STANAG 4404, IEEE Standard 1498, and other related documents. The SQA and SCM processes include these requirements as a routine part of their normal compliance assessment process. The software developers and testers will test generic or system-specific safety test requirements as a normal part of the software testing process. System Safety must review test cases and test procedures and make recommendations for additional or modified procedures and additional tests to ensure complete coverage of applicable requirements. However, review of test cases and procedures alone may not be sufficient. A number of the generic requirements require that the safety analyst ensure that the code meets the intent of the requirement versus the letter of the safety requirement. As noted earlier, due to the ambiguous nature of English language, specifications and safety requirements may be interpreted differently by the software developer thus not meeting the intent of the requirement. In some instances this requires an examination of the source code (see Section 4.3.7).

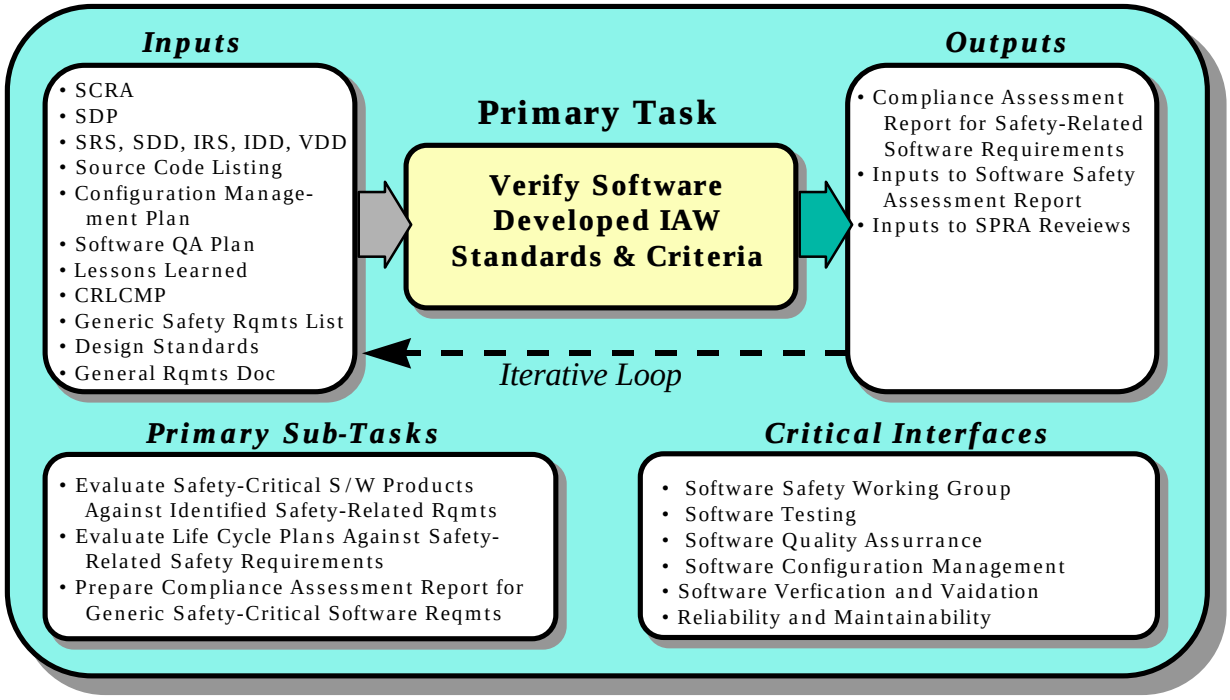


Figure 4-42: Software Requirements Verification

The generic software development requirements and guidelines are provided to the SQA team for incorporation into their plans, reviews, and assessment process. The Software Testing team generally assumes responsibility for incorporating generic test requirements into their test planning

process although many of the specific test requirements may be assigned to the individual software development teams for unit and integration testing. The configuration management team assumes responsibility for requirements related to configuration management and integrates them into the configuration management plans and processes. The latter include the participation by safety in the CM process.

The SSS Team reviews the assessment performed by the SQA team, incorporating the results for the safety related criteria into the final safety assessment. The assessment should occur on a real-time basis using representatives from the SSS Team (or a member of the SQA team assigned responsibility for software safety) to participate in the review of the software development process, code reviews, walk-throughs and peer reviews, to assess the degree of compliance. The assessment should include both the degree of compliance or the rationale for non-compliance with each of the criteria.

Software safety participates on a real-time basis with the configuration management process, therefore, the assessment against applicable criteria can occur during the process. To a large extent, the degree of compliance depends on the degree of involvement of system safety in the CM process and their thoroughness in fulfilling their roles.

The Compliance Assessment Report, as its name implies, is a compilation of the above compliance assessments with a final assessment as to the compliance of how the system complies with the applicable safety requirements. The compliance assessment is simply a portion of the overall safety assessment process used to ascertain the residual risk associated with the system.

4.4.4 SOFTWARE SAFETY RESIDUAL RISK ASSESSMENT

The safety risk assessment of software is not as straight forward a process as it is for hardware. Hardware safety assessment relies on hazard severity's and probabilities, whether qualitative or quantitative, coupled with the remaining conditions required to result in a hazardous condition or an accident. Qualitative probabilities are largely based on engineering judgment and experience with similar systems while quantitative probabilities are based on statistical measurements, such

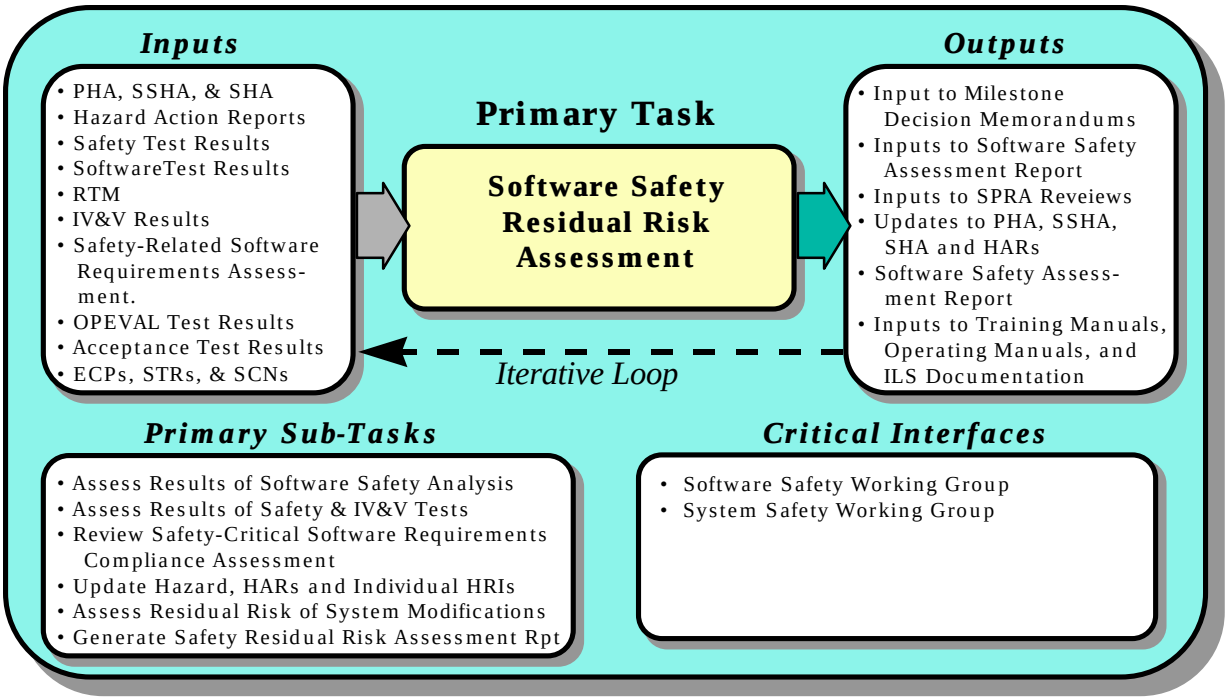


Figure 4-43: Residual Safety Risk Assessment

as reliability predictions. Residual risk is calculated using this information to compile a probability of a hazard occurring over the life of the system. Coupled with estimates of the likelihood of satisfying the remaining conditions that result in an accident or mishap, an estimate of the risk results. Unfortunately, reliability metrics for software are often meaningless, therefore, the use of qualitative risk assessment must be applied. The latter is based on an assessment by the analyst that sufficient analysis and testing have been performed to identify the hazards, develop and incorporate safety requirements into the design, and analyze and test their implementation to provide a reasonable degree of assurance that the software will have a sufficiently low level of risk. The Software Safety Assessment begins early in the system development process, largely starting with the aompliance assessment of the safety requirements discussed previously. However, the assessment cannot be completed until system level testing is complete. This includes operational test and evaluation, and the analyses which concludes that all identified hazards have been resolved. The software safety assessment process is generally complete when it is integrated with the Safety Assessment Report (SAR).

As described in Section 4.3.5, the analyst identifies the software causal factors early in the analytical phase and assigns a hazard severity and software control category to each. The result is a software hazard risk index for that causal factor. However, this is not a true measure of the safety risk but an indication of the degree of assurance required that the software will execute safely in the system context. It provides guidance on the amount of analysis and testing required to verify and validate the software associated with that function or causal factor. The software hazard risk index does not change unless the design is modified to reduce the degree of control the software exercises over the potentially hazardous function. However, analysis and testing performed on the software reduces the actual risk associated with it in the system application. In this manner, a qualitative hazard risk index may be assigned to the specific function based on

engineering judgment. The SSS Team needs to document these engineering judgments made for that function and its associated hazard(s) within the Hazard Control Records, if developed for that particular function, and update the analyses performed.

As with any hazards analysis, closure of the hazard requires that the analyst review the results of the analyses performed and the tests conducted at both a component and system level. Closure of hazards occurs on a real time basis as the design progresses. As this occurs, the SSS Team analyzes the design and implementation of the functions, both safety-critical and safety related, and determines whether the intent of the applicable safety design requirements is met and whether the implementation provides sufficient interlocks, checks, and balances, to ensure that the function will not contribute to a hazardous condition. Coupled with the results of testing performed on these functions, the analyst uses his or her best judgment as to whether the risk is sufficiently mitigated.

In performing the assessment of safety and IV&V testing, the software safety analyst must look at a variety of metrics associated with testing. These include path coverage, overall coverage of the program, and usage based testing. In general, testing that is limited to the usage base is inadequate for safety. Safety-critical modules often are those that execute only when an anomaly occurs. This results in a very low predicted usage and consequently, usage based testing performs very little testing on those functions. The net result is that anomalies may be present and undetected. The SSS Team should assess the degree of coverage and determine its adequacy. Generally, if the software testers are aware of the need for additional test coverage of safety-critical functions, they will be incorporated in the routine testing.

New requirements identified during the analysis and testing phases should be subjected to the same rigor as those in the original design. However, the SSS Team must pay particular attention to these areas since these are the areas most likely to contain errors in the latter stages of development. This is more a function of introducing requirements late, and reducing the time available for analysis and testing. In addition, the potential interactions with other portions of the system interfaces maybe unknown and may not receive the same degree of attention (especially at the integration testing level), as the original requirements.

The SSS Team must keep in mind throughout the safety assessment process, the ultimate definition of acceptable risk as defined by the customer. Where unacceptable or undesirable risks are identified, the SSS Team, in coordination with the SSWG, must provide the rationale for recommending to the customer and/or the safety review authority acceptance of that risk. Even for systems which comply with the level of risk defined by the customer's requirements, the rationale for that assessment and the supporting data must be provided. This material is also documented in the Safety Assessment Report.

The SAR contains a summary of the analyses performed and their results, the tests conducted and their results, and the compliance assessment described in section 4.4.3. Appendices to the SAR should include:

- The safety criteria and methodology used to classify and rank software related hazards (causal factors). This includes any assumptions made from which the criteria and methodologies were derived.
- The results of the analyses and testing performed.

- The hazards that have an identified residual risk and the assessment of that risk.
- The list of significant hazards and the specific safety recommendations or precautions required to reduce their risk.
- A discussion of the engineering decisions made that affect the residual risk at a system level.

The purpose of the SAR is to provide management an overall assessment of the risk associated with the software executing in the system context in an operational environment. The final section of the SAR should a statement by the Principal for Safety describing the overall risk associated with the software in the system context and their acceptance of that risk.

If the system will be presented to a safety review authority (SPRA), the software safety assessment must be presented in a rational, logical, and consistent manner. Ultimately, the SPRA provides the final safety assessment and risk acceptance based on the material presented by the system safety program.

4.5 MANAGING CHANGE

The old adage “nothing is constant except change” applies to software after the system is developed. Problems encountered during system level IV&V, and operational testing account for a small percentage of the overall changes. Problems or errors found by the users account for an additional percentage. However, the largest number of changes are the result of upgrades, updates, and pre-(or un)planned product enhancements. Managing change from a safety perspective requires that the SSS Team assess the potential impact of the change to the system. If the change is to correct an identified safety anomaly, or the change potentially impacts the safety of the system, the software systems safety assessment process must rely on the analyses and tests previously conducted.

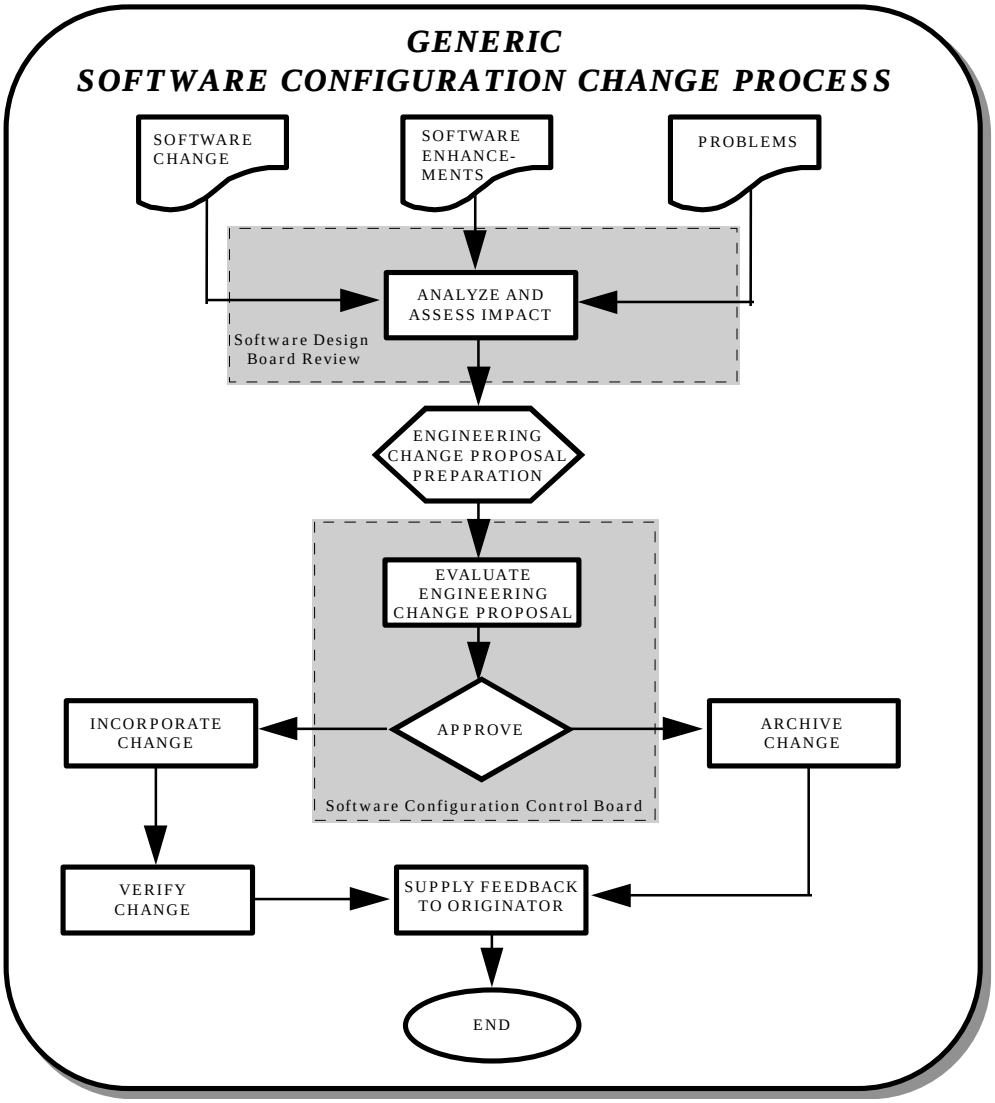


Figure 4-44: Generic Software Configuration Change Process

4.5.1 SOFTWARE CONFIGURATION CONTROL BOARD (CCB)

Configuration management is a system management function wherein the system is divided into manageable physical or functional configurations and grouped into Configuration Items (CI's). The management methods and techniques to control the design process involves the identification, control, status accounting, auditing and the Configuration Control Board (CCB). Configuration control (Figure 4-44) on the development process and products within that process are established once the system has been divided in to functional or physical configuration items. One purpose of the CCB is to divide the changes into Class I and Class II changes and to ensure that proper procedures are followed. The expressed purpose of this function is to ensure that project risk is not increased by the introduction of changes by unauthorized, uncontrolled, poorly coordinated or improper changes. These changes could directly or indirectly affect system safety and therefore require verification, validation and assessment scrutiny.

The CCB assists the program manager, design engineer, support engineers and other acquisition personnel in the control and implementation of Class I and Class II changes. Class I changes are those which affect form, fit, or function and require user concurrence prior to developer implementation. Class II changes are those not classified as Class I. Examples of Class II changes include editorial changes in documentation or material selection changes in hardware.

The CCB ensures that the proper procedures for authorized changes to the configuration item or related products or interfaces are followed and that risk is not increased by the change. The CCB should also ensure that any intermediate step which may halt, and expose the project to increased safety risk while halted, is controlled. The system safety assessment regarding a configuration change must include:

- Thorough review of the proposed change package (Engineering Change Proposal [(ECP)]) prepared by the engineer responsible for the change.
- Effects of the proposed change on subsystem and system hazards previously identified. This to include existing or new functional, physical, or zonal interfaces.
- Determination as to whether the proposed change introduces new hazards to the system or to its operations and support functions.
- Determination as to whether the proposed change circumvents existing (or proposed) safety systems.
- Analysis of all hardware/software and system/operator interfaces.

The SSS Team follows much the same process, on a smaller scale, as they followed during system development. The analyses will have to be updated and appropriate tests reaccomplished, particularly the safety tests related to those portions of the software being modified. The development of the change follows the engineering change proposal process complete with the configuration control process. The overall process follows through and concludes with a final safety assessment of the revised product. It is important to remember that some revalidation of the safety of the entire system may be required depending on the extent of the change. One very important aspect of managing change is the change in system functionality. This includes the addition of new functionality to a system that adds safety-critical functions to the software. For example, if the software developed for a system did not contain safety-critical functions in the original design yet the modifications add new functionality that is safety-critical, the software safety effort will have to revisit a great deal of the original software design to assess its safety risk potential. The software safety analysts will have to revisit both the generic safety design requirements and the functionally derived safety requirements to determine their applicability in light of the proposed software change. Where the tailoring process determined that certain generic requirements were not applicable, the rationale will have to be examined and the applicability re-determined. It is very easy for the procuring agency to try to argue that the legacy software is safe and the new functionality requires that it be the only portion examined. Unless very high quality software engineering standards were followed during the original development, it will be very difficult for the safety analyst to ensure that the legacy software cannot adversely impact the new functionality. Again, the process used is much the same as it was for the original software development.

4.6 REUSABLE SOFTWARE

Reusable software, as its name implies, is software from a previous program that is reused to reduce development costs. The software may or may not meet all of the needs of the system, or it may include functionality not required by the system under development, therefore requiring modification. In the latter case, the software is generally not modified to remove the unnecessary functionality. The software safety analyst must analyze the software proposed for reuse in the system context to determine what its role and functionality will be in the final system. The analysis must be performed regardless of whether or not the software was safety "qualified" in its previous application since the safety-criticality of the software is application specific. Thus, the reuse of software does not ease the software safety program burden.

4.7 COTS SOFTWARE

The safety assessment of Commercial Off the Shelf (COTS) software poses one of the greatest challenges to the safety certification of systems. COTS software is generally developed for a wide range of applications in the commercial market. The software is developed to an internal company standard or to an industry standard, such as IEEE, ANSI, or NIST. In general, the language used is determined by the company or the individual project team. Since the vendor releases only compiled versions of the product, there is often no way to determine which language is used. Because the developer can only guess at the applications that the software may be used in, specific issues related to application are often not addressed during the design. However, attempts are made to ensure that the software will be compatible with potential system and software configurations to which it must interface. An excellent example of such an issue occurred with the Red Cross Blood Databank. The databank used a COTS database manager and a COTS network system tailored to the Red Cross's needs. The databank stores information regarding individual units of blood including whether or not the blood has any infectious diseases. The databank operated for several years when a problem occurred. Two separate laboratories were used to test blood type and check for diseases. However, each laboratory had responsibility for different diseases. The incident occurred when both laboratories accessed the same record simultaneously to enter their results. Laboratory A tested for HIV in the blood samples, noted the results in the database record and then saved it. Meanwhile, laboratory B discovered infectious hepatitis in a blood sample, noted the fact in the same record and saved it, overwriting laboratory A's data. Infectious hepatitis can be destroyed by freezing methods, therefore the Red Cross believed the blood safe for use after processing. Thus, whether the blood was infected with the HIV was unknown. Fortunately, the error was discovered prior to distribution of the blood.

The Red Cross example illustrates how the application of a program affects safety issues. The database program designer could not anticipate all applications of the program and designed it such that it would preclude such an event. Likewise, the network program designer could not anticipate such an issue either. Therefore, he could not anticipate that two users accessing a record simultaneously would lead to a hazard in an unspecified system. The safety of COTS software in other systems, likewise, depends upon the application. A specific hazard depends on the application and the system design. Likewise, COTS software used for networking or other shell functions, can affect critical software and cause unpredictable results.

In the conduct of a system safety program, the SSS Team analyzes the system to identify the undesired events (mishaps) and the potential causes that may lead to those mishaps (hazards). By eliminating the hazards or reducing the probability of their occurrence, the analyst reduces the overall safety risk associated the system. However, to eliminate or reduce the risk, the safety analyst requires detailed knowledge of the system design and use. Software safety has three goals:

- First, to ensure that the software contains no constructs that could result in a hazard.
- Second, to ensure that hardware or software components cannot fail in such a manner that they propagate through the software and result in a hazard.
- And third, that the software is used where practical to mitigate other hazards in the system.

Accomplishing the first goals requires that the safety engineer analyze the software in the context of its application to the system. The second goal requires that the safety engineer participate in the design at the system and software levels and have detailed documentation on the software available for analysis. The third goal requires that the safety engineer identify the hazards and make recommendations for functions that will to reduce the risk of other identified hazards.

COTS software poses several problems in this regard. Generally, only user documentation is available from commercial software developers. The type of documentation necessary to conduct a detailed analyses is usually not available. The developer may not generate high level specifications, functional flow diagrams, data flow diagrams, or detailed design documents for a given commercial software package. The software safety engineer can only trace system hazard causal factors to the COTS software, but possesses no ability to develop and implement design requirements within the software to reduce the risk of other hazards associated with the system.

Configuration management also becomes a significant issue in safety-critical applications of COTS software. If the design agent performs sufficient analysis and testing to verify that a specific version of a COTS program will not adversely affect the safety of the system, the certification applies only to the specific version of the COTS program tested. Unfortunately, the user does not have information on when changes are inserted into the COTS. Therefore, unless only the tested version is released to the user and used for future releases of the applications software, the safety certification for that software is lost. Finally, the ability to determine whether or not hardware or software failures can propagate through the COTS software and result in a hazard is limited to the amount of testing that can be conducted.

Testing of the COTS software in this application is very limited in its ability to provide evidence that the software cannot influence system hazards. Testing in a laboratory cannot duplicate the operational environment nor can it duplicate every possible combination of events. Even when the design and implementation of the software is known in detail, many constraints still apply even though test engineers can develop procedures to test software paths, based on their knowledge of failures and operational errors of the software design, the testing organization, like the safety organization, must still treat COTS software as a "black box”, developing tests to measure the response of the software to input stimulus. Hazards identified through "black box" testing are often happenstance and sometimes difficult to duplicate. Timing issues and data senescence issues also are difficult to fully test in the laboratory environment even for software of a known design.

Without the ability to analyze the code, determining potential timing problems in the code is difficult at best. Without detailed knowledge of the design of the software, the system safety and test groups can only develop limited testing to verify the safety and fail-safe features of the system.

Like commercially available software, commercially available hardware also possesses limitations that make safety certification of software applications difficult in most cases. Commercial systems, such as workstations, are designed to industry standards which are much less stringent than the military standards. Any number of vendors may supply parts or subsystems, with little control over the quality or conformance to requirements other than those imposed by the developer. As with COTS software, the vendor generally does not document the firmware embedded in the off-the-shelf hardware in a manner that can be analyzed by the user. In addition, high level and detailed documentation on the hardware design may not be available for review and analysis. Therefore, it is easy to draw corollaries between system safety issues related to COTS software, and COTS hardware. Open architecture systems employing COTS software and NDI hardware pose additional safety concerns requiring different solutions than existing system architectures require.

One method of reducing the risk associated with COTS software applications is to reduce its influence on the safety-critical functions in the system. This requires isolation (i.e. fire walling) of the COTS software from the safety-critical functions. Unfortunately, in many applications, (such as communications with a local area network, operating systems, and operating environments), the software may still have the ability to adversely affect safety-critical functionality. In these cases, additional safeguards must be used. Isolation of safety-critical functions from the COTS software requires application-specific software on either side of the COTS software that performs the necessary sanity checks to preclude hazards' occurrence. The risk associated with COTS and NDI must be considered during the PHA phase of the program. The identification of potential causes that COTS & NDI may have on the system hazards may help to identify additional software/system safety requirements. Testing at the functional and system levels will also help to mitigate the risk associated with COTS and NDI. However, the risk associated with testing is that it is impossible to test all possible paths, conditions, timing, and data senescence problems in a test environment. It is recommended that the testers develop a set of benchmark tests that test the full spectrum of an applications functionality that can be analyzed and executed for each revision.